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** **1**

**2 Platform** **1**

**3 Code** **1**

**4 Output** **5**

**5 Conclusion** **5**

# 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++)
    {
```

```
34          if (pow(2, i) >= int(m) + i + 1)
35          {
36              r = i;
37              break;
38          }
39      }
40      ::r_val = r;
41      return int(m) + r;
42  }
43
44  // Converts Binary array into decimal
45  void convert_binary_to_decimal(bool parity[])
46  {
47      int decimal = 0;
48      for (int i = 0; i < ::r_val; i++)
49      {
50          if (parity[i])
51          {
52              decimal += pow(2, i);
53          }
54      }
55      if (decimal)
56      {
57          ::error_bit = decimal;
58      }
59  }
60
61  // Fills the values with r_array in the 2d array
62  void fill_r_values(int hamming_len)
63  {
64      int count;
65      bool should_add;
66      for (int k = 0; k < hamming_len; k++)
67      {
68          count = 0;
69          should_add = false;
70          for (int i = 0, j = 1; i <= hamming_len; i++)
71          {
72              if (count == pow(2, k))
73              {
74                  count = 0;
75                  should_add = !should_add; // flips it
76              }
77
78              if (should_add)
79              {
80                  r_array[k][j] = i;
81                  j++;
82              }
83              count++;
84          }
85      }
86  }
87
88  // Fills the first column of the r_array table, to 1 or 0 for maintaining even
        parity.
89  void fill_r_parity(int hamming_len, const bool hamming[])
90  {
91      int count;
```

```
92      bool parity;
93      for (int i = 0; i < ::r_val; i++)
94      {
95          // check parity
96          count = 0;
97          for (int j = 2; j <= (hamming_len / 2) + 1; j++)
98          {
99              hamming[r_array[i][j] - 1] ? count++ : count;
100         }
101         parity = count % 2 != 0; // if number of 1's is even
102         r_array[i][0] = parity;  // assign parity bit
103     }
104 }
105
106 // Fills the hamming array by looking at the parity r_array bits from the r_array
        array.
107 void fill_hamming(int hamming_len, bool hamming[])
108 {
109     int k = 0;
110     for (int j = 0; j < hamming_len; j++)
111     {
112         if (j == pow(2, k) - 1)
113         {
114             hamming[j] = r_array[k][0];
115             k++;
116         }
117     }
118 }
119
120 void display_hamming(int hamming_len, bool hamming[])
121 {
122     for (int i = hamming_len - 1; i >= 0; i--)
123     {
124         cout << hamming[i];
125     }
126     cout << endl;
127 }
128
129 // This function does the entire error correction, and prints the process as well
130 void detect_errors(int hamming_len, bool hamming[50])
131 {
132     int count;
133     bool parity[::r_val];
134
135     // Display new hamming code with flipped bit, and the old one as well.
136
137     // Deduce values of r_array from the new hamming code
138
139     // from the previous r_array table that we already have,
140     for (int i = 0; i < ::r_val; i++)
141     {
142         // check parity
143         count = 0;
144         for (int j = 1; j <= hamming_len / 2 + 1; j++)
145         {
146             hamming[r_array[i][j] - 1] ? count++ : count;
147         }
148         parity[i] = count % 2 != 0; // if number of 1's is even
149     }
```

```
150     // converted parity bits to decimal, and then find the flipped bit
151     convert_binary_to_decimal(parity);
152 
153     // Display the flipped bit and then the corrected hamming code, with the
        original hamming code.
154     cout << "The Bit which was changed is: " << ::error_bit << endl;
155     cout << "The Hamming code with the correction is: " << endl;
156     hamming[::error_bit - 1] = !hamming[::error_bit - 1];
157     display_hamming(hamming_len, hamming);
158 }
159 
160 int main()
161 {
162     string input;
163     int hamming_len, flipped_bit = 0;
164     // Input the value as a string, as we don't know how long it can be.
165     cout << "Enter the Input : " << endl;
166     cin >> input;
167 
168     // Edge Case
169     if (input.length() == 0)
170         return 0;
171     else
172         m = input.length();
173 
174     // Find the value of r_array and the length of the hamming code
175     hamming_len = calc_length(input);
176 
177     // Declare an array to store the hamming code
178     bool hamming[50] = {};
179 
180     // Store the bits
181     for (int i = 0, j = 0, k = int(m); i < hamming_len; i++)
182     {
183         if (i != (pow(2, j) - 1))
184         {
185             hamming[i] = (input[k - 1] == '1');
186             k--;
187         }
188         else
189         {
190             j++;
191         }
192     }
193 
194     // fill the values of r_array till hamming_len
195     fill_r_values(hamming_len);
196 
197     // Fill r_array with even parity
198     fill_r_parity(hamming_len, hamming);
199 
200     // Fill the hamming code
201     fill_hamming(hamming_len, hamming);
202 
203     cout << "The Hamming code to be sent by the sender is: " << endl;
204     display_hamming(hamming_len, hamming);
205 
206     // Implement Error Detection
207 
```

```
208     cout << "What bit would you like to flip? (Starting from 1, from right)" <<
        endl;
209     cin >> flipped_bit;
210     // Changing the Hamming code
211     hamming[flipped_bit - 1] = !hamming[flipped_bit - 1];
212     cout << "The Hamming code after the error is: " << endl;
213     display_hamming(hamming_len, hamming);
214 
215     cout << "Now Calculating Error" << endl;
216     detect_errors(hamming_len, hamming);
217 
218     return 0;
219 }
```

Listing 1: Hamming Code.cpp

## 4 Output

```
1  Enter the Input :
2  11011011011
3  The Hamming code to be sent by the sender is:
4  110110111011110
5  What bit would you like to flip? (Starting from 1, from right)
6  3
7  The Hamming code after the error is:
8  110110111011010
9  Now Calculating Error
10 The Bit which was changed is: 3
11 The Hamming code with the correction is:
12 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.