Aplikasi *Hamming Code* sebagai *Error Correcting Code* pada Transmisi Data Suara

Kandida Edgina Gunawan - 13521155[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521155@std.stei.itb.ac.id

***Abstract*—** *Error Correcting Code* adalah suatu skema yang digunakan untuk menangani adanya perubahan representasi biner pada data saat dilakukannya transmisi data. Perubahan representasi biner ini dapat terjadi karena adanya beberapa faktor seperti adanya interferensi gelombang, lonjakan listrik, kebisingan listrik, dan masih banyak lagi. Perubahan representasi biner pada suatu data dapat menyebabkan berubahnya informasi yang ingin disampaikan dari data tersebut. Oleh karena itu, diperlukan sebuah algoritma untuk mengatasinya. Salah satu jenis *Error Correcting Code* adalah *Hamming Code* yaitu suatu algoritma yang dapat mendeteksi perubahan bit pada data, namun *Hamming Code* hanya efektif untuk diimplementasikan jika data yang berubah hanya 1 bit saja. Pada makalah ini, akan dibahas lebih lanjut mengenai implementasi *Hamming Code* yang dikhususkan pada transmisi data bertipe audio.

***Keywords*—** *Error Correcting Code*, *Hamming Code*, audio, data

I. PENDAHULUAN

Data merupakan salah satu aspek yang tidak bisa lepas dari kehidupan manusia. Setiap hari, terjadi pertukaran data antara tiap individu. Bentuk data yang disebarkan pun ada beraneka ragam, dari data berbentuk visual seperti gambar, berbentuk audio seperti musik atau rekaman suara, ataupun dalam bentuk teks biasa. Dalam dunia *computing*, semua data baik itu dalam bentuk gambar, audio, ataupun lainnya akan diubah ke dalam representasi biner yang dapat dimengerti oleh mesin. Setelah diubah ke dalam representasi biner, barulah suatu data dapat diproses oleh mesin.

Dalam proses transmisi dan pertukaran data, tak jarang ditemui, representasi biner dari data tersebut dapat berubah. Hal ini dapat disebabkan oleh beberapa faktor, yaitu karena adanya interferensi gelombang, lonjakan listrik, kebisingan listrik, tergoresnya *compact disk*, dan masih banyak lagi faktor-faktor eksternal lain yang tidak dapat diprediksi. Berubahnya representasi biner dari suatu data dapat menimbulkan dampak yang fatal. Perubahan representasi biner dari suatu data dapat membuat informasi yang ingin disampaikan pengirim kepada penerima menjadi berubah. Oleh karena itu, dibutuhkan suatu algoritma yang dapat mengatasi perubahan representasi biner dari suatu data pada proses transmisi data ini.

Algoritma *Hamming Code* merupakan salah satu algoritma yang dapat digunakan untuk mendeteksi adanya perubahan representasi biner dari sebuah data. Algoritma ini tak hanya dapat mendeteksi adanya perubahan, namun juga dapat menemukan lokasi bit yang berubah. Hal ini tentunya memudahkan kita untuk mengembalikan nilai bit data yang sebenarnya pada posisi yang tepat.

Pada makalah ini, akan dibahas lebih lanjut mengenai implementasi dari *Hamming Code* untuk mengatasi perubahan representasi biner pada data. Pada makalah ini, jenis data yang ingin dipakai adalah data bertipe audio. *Hamming Code* ini nantinya akan digunakan untuk memeriksa representasi biner dari hasil ekstraksi fitur-fitur audio. Harapan penulis dari menulis makalah ini adalah penulis dapat lebih memahami bahwa dengan adanya algoritma *Hamming Code* ini, proses transmisi data bukan hanya menjadi lebih efisien saja, tapi juga menjadi lebih akurat.

II. LANDASAN TEORI

A. Aljabar Boolean

1. Definisi

George Boole, pencetus aljabar boolean, menyatakan bahwa himpunan dan logika proposisi mempunyai sifat-sifat yang serupa. Aturan-aturan dasar logika yang dikemukakan dalam buku The Laws of Thought membentuk struktur matematika yang disebut sebagai aljabar boolean.

Misalkan B adalah suatu himpunan yang didefinisikan pada dua operator biner, + dan ., dan sebuah operator uner, ’. Misalkan 0 dan 1 adalah dua elemen yang berbeda dari B.

Disebut aljabar Boolean jika untuk setiap a, b, c ∈ B berlaku aksioma berikut:

(I) Indentitas
1) $a + 0 = a$
2) $a \,.\, 1 = a$

(II) Komutatif
1) $a + b = b + a$
2) $a \,.\, b = b \,.\, a$

(III) Distributif
1) $a \,.\, (b + c) = (a \,.\, b) + (a \,.\, c)$
2) $a + (b \,.\, c) = (a + b) \,.\, (a + c)$

(IV) Komplemen
Untuk setiap $a \in B$ terdapat elemen unik $a' \in B$ sehingga
1) $a + a'$ = 1
2) $a \,.\, a'$ = 0

2. Aljabar Boolean 2 Nilai

Pada aljabar 2-nilai :

(I) $B = \{0, 1\}$,

(II) Operator biner: + dan ., operator uner: ’

(III) Kaidah untuk operator biner dan operator uner:

a	b	$a \,.\, b$	$a + b$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

a	a'
0	1
1	0

Terdapat pula operator lain dari aljabar boolean yang dapat pula diturunkan dengan kombinasi operator '+' dan '.' yaitu operator XOR ($\oplus$).

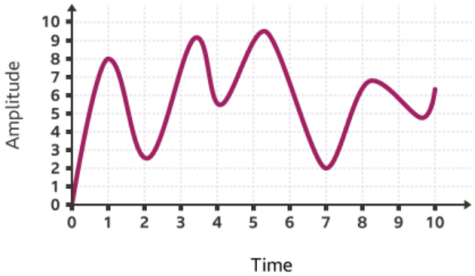
$$a \oplus b = a'b + ab' = (a + b) \,.\, (a' + b')$$

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

B. Bunyi

Setiap data yang diproses di dalam komputer, baik itu dalam bentuk teks, gambar, maupun suara, akan direpresentasikan dalam bilangan biner. Setiap digit dari bilangan biner merepresentasikan memori sebesar 1 bit yang digunakan. Bit adalah satuan terkecil dari data yang digunakan di dalam komputer.

Bunyi (sound) adalah suatu gelombang tekanan (pressure wave), yaitu gelombang yang mana gangguan yang disebarkan adalah variasi tekanan di dalam mediumnya (dalam hal ini, mediumnya adalah udara). Ketika seseorang berbicara atau ketika kita memainkan alat musik, terjadi perubahan secara periodik pada tekanan udara yang kita interpretasikan sebagai bunyi. Untuk mengubah data dalam bentuk bunyi ke dalam representasi biner, data bunyi yang dimiliki akan diubah terlebih dahulu menjadi sinyal digital. Konverter analog-digital akan menangkap gelombang suara pada interval waktu tertentu dan hasil rekaman yang ditangkap ini disebut sebagai sampel. Pada proses pengambilan sampel, dihitung berapa banyak tekanan berubah dalam interval waktu tertentu. Sebagai contoh, gelombang bunyi seperti di bawah ini dapat diambil sampelnya di setiap titik sampel waktu:



Gambar 1. Sampel gelombang bunyi
Sumber : https://www.bbc.co.uk/bitesize/guides/zfspfcw/revision/9

Bunyi yang telah diambil sampelnya pada tiap waktu tertentu kemudian akan diambil nilai amplitudonya dan dihitung representasi binernya

Sample	1	2	3	4	5	6	7	8	9	10
Denary	8	3	7	6	9	7	2	6	6	6
Binary	1000	0011	0111	0110	1001	0111	0010	0110	0110	0110

Gambar 2. Representasi Biner dari Amplitudo Sampel
Sumber : https://www.bbc.co.uk/bitesize/guides/zfspfcw/revision/9

Sample rate adalah banyaknya sampel yang direkam dalam suatu periode waktu tertentu. Semakin tinggi sample rate yang dimiliki, semakin dekat pula sinyal yang direkam dengan sinyal aslinya. Sample rate diukur dalam hertz. Akan tetapi, tingginya sample rate akan berbanding lurus dengan besarnya file. Akibatnya, file suara seringkali merupakan kompromi antara kualitas dan ukuran file. File audio biasanya direkap pada 44.1kHz. Ini cukup tinggi untuk menghasilkan kualitas suara yang baik, namun tetap menjaga ukuran file yang dihasilkan agar tidak terlalu besar.

Bit depth merupakan banyaknya bit yang digunakan untuk merekam tiap sampel bunyi. Semakin tinggi bit depth yang digunakan, semakin akurat pula suara yang terekam, namun ukuran file yang dihasilkan juga akan semakin besar. Biasanya bit depth berada pada kisaran 16 bit dan 24 bit.

Bit rate adalah ukuran banyaknya data yang dapat diproses untuk tiap detik bunyi. Cara menghitung bit rate adalah sebagai berikut:

$$Bit\ rate = sample\ rate \times bit\ depth$$

C. Hamming Code

Hamming Code adalah salah algoritma *error correction* yang digunakan untuk mendeteksi dan melakukan koreksi pada kesalahan yang ditimbulkan akibat perpindahan atau penyimpanan data dari pengirim ke penerima. Teknik ini dikembangkan oleh R. W. Hamming. Ide dari Hamming Code adalah menggunakan bit tambahan pada informasi yang disimpan. Bit bawaan pada tiap informasi ini yang akan memastikan tidak ada bit yang hilang atau nilainya berubah selama proses transfer data. Jumlah bit tambahan yang digunakan dapat dihitung dengan persamaan berikut,

$$2^r \geq m + r + 1$$

Dengan r adalah jumlah bit tambahan yang akan digunakan dan m adalah jumlah bit data sebenarnya yang ingin dikirim.

Istilah lain yang juga penting untuk diketahui adalah bit paritas. Bit paritas adalah bit yang ditambahkan pada bit data yang sebenarnya ingin diproses untuk memastikan banyak bit bernilai 1 pada data adalah ganjil atau genap. Terdapat 2 jenis bit paritas:

a) Bit paritas genap

Pada bit paritas genap, banyak bit data yang bernilai 1 akan dihitung jumlahnya. Apabila banyak bit data yang bernilai 1 merupakan suatu bilangan ganjil, nilai dari bit paritas akan menjadi 1. Apabila banyak bit data yang bernilai

1 merupakan suatu bilangan genap, nilai bit paritas akan diatur menjadi 0.

b) Bit paritas ganjil
Pada bit paritas ganjil, banyak bit data yang bernilai 1 akan dihitung jumlahnya. Apabila banyak bit data yang bernilai 1 merupakan suatu bilangan ganjil, nilai dari bit paritas akan menjadi 0. Apabila banyak bit data yang bernilai 1 merupakan suatu bilangan genap, nilai bit paritas akan diatur menjadi 1.

Ide umum dari algoritma *Hamming Code* adalah menggunakan bit paritas untuk mengidentifikasi adanya kekeliruan pada bit data. Alur berpikir dari *Hamming Code* adalah sebagai berikut.

1. Setiap posisi bit yang merupakan hasil perpangkatan dari 2, yaitu 1, 2, 4, 8, …, merupakan posisi dari bit paritas
2. Posisi bit selain posisi bit paritas merupakan posisi bit data yang sebenarnya.
3. Setiap bit data termasuk ke dalam set bit paritas yang unik, tergantung pada posisi bit data tersebut jika dituliskan di dalam representasi biner. Bit paritas 1 meliputi semua bit yang posisi bitnya jika dituliskan dalam representasi biner mengandung nilai 1 pada posisi bit terakhirnya misalnya 1, 3, 5, 7, 9, …
   Begitu pula dengan bit paritas 2 yang meliputi semua bit yang posisi bitnya jika dituliskan dalam representasi biner mengandung nilai 1 pada posisi bit kedua dari belakang, misalnya 2, 3, 6, 7,…
4. Setelah mengelompokkan berdasarkan paritasnya, dicek banyaknya kemunculan ‘1’ pada tiap paritas. Jika pada paritas 1, banyak kemunculan ‘1’ merupakan suatu bilangan ganjil, nilai bit paritas 1 haruslah ‘1’, begitu juga sebaliknya, jika pada paritas 1, banyak kemunculan ‘1’ merupakan bilangan genap, nilai bit paritas haruslah ‘0’. Hal ini juga berlaku untuk semua bit paritas lainnya.
5. Perubahan nilai pada bit data dapat diidentifikasi dengan memeriksa bit paritasnya. Jika terdapat perubahan pada bit paritas, dapat dipastikan bahwa bit datanya telah mengalami perubahan.

III. PEMBAHASAN

A. Ekstraksi Fitur Bunyi

Berdasarkan penjelasan pada landasan teori, sampel yang akan diambil adalah amplitudo bunyi dalam periode waktu tertentu. Untuk mengekstraksi fitur-fitur dari bunyi ini, digunakan beberapa *library librosa* dan *Ipython.display* dari *python* yang memang memiliki banyak fungsi bawaan untuk mengolah data berupa *audio*. Selain itu, digunakan pula beberapa library tambahan seperti numpy dan juga library buatan sendiri, yaitu hamming_code yang dibuat untuk mempermudah implementasi program.

```
import librosa
import librosa.display
import matplotlib.pyplot as plt
from IPython.display import Audio
from hamming_code import *
```

Gambar 3.1. Library yang digunakan di dalam program

Dalam implementasi program kali ini, digunakan file audio yang terdapat pada link *google drive* berikut: https://drive.google.com/file/d/1NbXrWmJyCEE-p7H1fSoH6rlnVYm3kfXz/view?usp=sharing

Untuk mengekstraksi sampel bunyi tersebut digunakan *code* sebagai berikut.

```
# Original audio file
audio_file = 'download.wav'
y, sr = librosa.load(audio_file)
Audio(y, rate = sr)
```

Gambar 3.2. *Code* ekstraksi fitur audio

y merupakan array yang berisi fitur hasil ekstraksi file audio Berikut merupakan hasil ekstraksi dari file audio ‘download.wav’

```
[-2.0446777e-03 -6.4086914e-04 -5.7983398e-04 ...  0.0000000e+00
 -3.0517578e-05  0.0000000e+00]
```

Gambar 3.3. Hasil ekstraksi fitur audio

Code Audio(y, rate = sr) digunakan sebagai perintah pada ipynb untuk memutar audio dengan hasil ekstraksi y.

B. Identifikasi Error dengan Hamming Code

Hamming Code sangat berguna untuk memeriksa adanya kekeliruan pada representasi biner sebuah data. Kendati demikian *Hamming Code* hanya dapat memeriksa kekeliruan biner pada satu bit data saja seperti yang sudah dijelaskan pada landasan teori di atas.

Untuk memeriksa adanya *error* pada representasi biner, serta memberikan koreksi kepada representasi biner tersebut, dibuat beberapa fungsi yang dapat membantu implementasi dari *hamming code* sebagai berikut.

```
def num_of_redundant(data_bits):
    redundant = 0
    while ((2**redundant) < (data_bits+redundant+1)):
        redundant += 1
    return redundant
```

Gambar 3.4. Fungsi num_of_redundant

Fungsi num_of_redundant di atas berguna untuk menghitung berapa banyak bit yang harus ditambahkan pada data untuk melakukan identifikasi *error* pada representasi biner data tersebut.

```python
def redundant_position(data, redundant_bits):
    power = 0
    index = 1
    result = ''
    for i in range(1, (len(data) + redundant_bits + 1)):
        if (i != (2**power)):
            result += data[-1*index]
            index += 1
        else:
            result += '0'
            power += 1
    return result[::-1]
```

Gambar 3.5. Fungsi redundant_position

Fungsi redundant_position di atas berfungsi untuk menempatkan bit-bit tambahan pada data dengan nilai awal bit-bit tambahan tersebut adalah 0.

```python
def parity_bits(new_data, redundant_bits):
    n = len(new_data)
    for i in range(redundant_bits):
        parity = 0
        for j in range(1, n+1):
            if ((2**i) & j == (2**i)):
                parity ^= int(new_data[-1*j])
        new_data = new_data[:n-(2**i)] + str(parity) + new_data[n-(2**i)+1:]
    return new_data
```

Gambar 3.6. Fungsi parity_bits

Fungsi parity_bits di atas berfungsi untuk memberikan nilai pada bit-bit paritas yaitu bit-bit yang berada pada posisi perpangkatan dari dua (posisi 1, 2, 4, 8, …) sesuai dengan aturan bit paritas genap.

```python
def error_detection(new_data, redundant_bits):
    n = len(new_data)
    datacheck = ''
    for i in range(redundant_bits):
        parity = 0
        for j in range(1, n+1):
            if ((2**i) & j == (2**i)):
                parity ^= int(new_data[-1*j])
        datacheck = str(parity) + datacheck
        print(datacheck)
    return int(datacheck, 2)
```

Gambar 3.7. Fungsi error_detection

Fungsi error_detection ini digunakan untuk memeriksa dan membandingkan kembali bit-bit paritas dengan bit-bit data yang telah disimpan. Jika ternyata tidak terdapat bit data yang berubah, fungsi akan mengembalikan nilai 0.

```python
def correcting_error(new_data, errorpos):
    n = len(new_data)
    error = new_data[errorpos-1]
    res = ''
    for i in range(n):
        if (i == (errorpos-1)):
            if (error == '0'):
                res += '1'
            else:
                res += '0'
        else:
            res += new_data[i]
    return res
```

Gambar 3.8. Fungsi correcting_error

Fungsi correcting_error ini akan mengubah bit data yang berubah dengan nilai yang seharusnya.

Selain fungsi-fungsi di atas, terdapat juga beberapa fungsi yang digunakan untuk mengubah bentuk data *float* ke dalam bentuk representasi biner ataupun sebaliknya, yaitu mengubah bentuk representasi biner ke dalam data bertipe *float*. Fungsi-fungsi ini dibuat dengan melihat bahwa nilai amplitudo pada bunyi yang tak bertipe *integer*, namun bertipe data *float*.

```python
def dec2bin(number):
    ans = ''
    if (number == 0):
        return 0
    while (number):
        ans += str(number & 1)
        number = number >> 1

    ans = ans[::-1]
    return ans
```

Gambar3. 9. Fungsi untuk mengubah *integer* ke dalam representasi biner (bertipe *string*)

```python
def float_to_binary(number, places):
    integer, dec = str(number).split('.')
    if (integer == ''):
        integer = 0
    integer = int(integer)
    dec = number - integer
    decbin = ''
    val = -999
    while (places and val != 1):
        dec *= 2
        if (dec > 1):
            decbin += '1'
            dec -= 1
        elif (dec == 1):
            decbin += '1'
            val = 1
        else:
            decbin += '0'
        places -= 1

    int_temp = dec2bin(integer)
    result = int_temp
    result = result + '.' + decbin
    return result, int_temp, decbin
```

Gambar 3.10. Fungsi untuk mengubah tipe data float ke dalam representasi binernya dalam *string*

```python
def bin_to_float(intbin, decbin):
    result = 0
    n1 = len(intbin)
    integer = 0
    for i in range(1, n1+1):
        if (intbin[-1*i] == '1'):
            integer += (2**(i-1))
    n = len(decbin)
    temp = 0
    for i in range(1, n+1):
        if (decbin[-1*i] == '1'):
            temp += 1
            temp /= 2
        else:
            temp /= 2
    result = integer + temp
    return result
```

Gambar 3.11. Fungsi untuk mengubah representasi biner ke

data bertiper *float*

C. Implementasi Hamming Code pada Data Audio

Alur implementasi dari program yang dibuat adalah sebagai berikut.

1. Dilakukan proses ekstraksi fitur-fitur audio pada *file* audio yang ingin digunakan
2. Mengubah tiap data fitur-fitur audio hasil ekstraksi dengan jumlah bit yang diubah tepat 1 untuk tiap fitur audio. Hasil ekstraksi yang fitur-fitur audionya telah diubah kemudian disimpan pada sebuah variabel.
3. Memeriksa data fitur-fitur audio yang telah diubah nilainya sebelumnya. Proses pemeriksaan dilakukan dengan menggunakan *Hamming Code* untuk mengidentifikasi letak bit *error* dan melakukan *correction* secara bersamaan. Fitur-fitur audio yang telah melalui proses koreksi ini kemudian disimpan ke dalam suatu variabel yang lain.
4. Dilakukan proses pembandingan *file audio* sebenarnya dengan *audio* yang memiliki 1 bit *error* dan audio yang sudah melalui proses koreksi dengan menggunakan *Hamming Code*.

```
# Audio file with 1 bit error
a = y
for i in range(len(a)):
    a[i] = a[i].round(5)
    if(a[i] < 0.0001):
        a[i] = 0.0

    integer, dec = str(a[i]).split('.')

for i in range(len(a)):
    binary, intbin, decbin = float_to_binary(a[i])
    if(intbin == ''):
        intbin = '0'
    lint = len(intbin)
    ldec = len(decbin)
    temp = intbin + decbin
    t1 = ''
    for i in range(len(temp)):
        if(i == 1):
            t1 += '0'   #always put 0 at position i (can cause 1 bit err
        else:
            t1 += temp[i]
    newint = '0'
    newdec = ''
    for i in range(len(t1)):
        if(lint > 0):
            newint += t1[i]
            lint-=1
        else:
            newdec += t1[i]
    result = bin_to_float(newint, newdec)
    a[i] = result
Audio(a, rate = sr)
```

Gambar 3.12. Proses mengubah data hasil ekstraksi audio menjadi data yang memiliki kekeliruan sebanyak 1 bit

```
final = a
for i in range(len(y)):
    y[i] = y[i].round(5)
    if(y[i] < 0.0001):
        y[i] = 0.0
for i in range(len(final)):
    binary, intbin, decbin = float_to_binary(y[i])
    m = len(intbin) + len(decbin)
    redundant = num_of_redundant(m)
    arr = parity_bits(arr, redundant)
    binary1, intbin1, decbin1 = float_to_binary(final[i])
    lenint = len(intbin1)
    lendec = len(decbin1)
    temp = intbin1 + decbin1
    correction = error_detection(temp, redundant)
    if(correction != 0):
        errorpos = len(arr)-correction+1
        temp = correcting_error(arr, errorpos)
        intbin1 = ''
        decbin1 = ''
        for i in range(len(temp)):
            if(i <= lenint-1):
                intbin1 += temp[i]
            else:
                decbin1 += temp[i]
        final[i] = bin_to_float(intbin1, decbin1)

Audio(final, rate = sr)
```

Gambar 3.13. Proses memeriksa perubahan representasi biner pada data dan *correction error*

Pada proses-proses di atas, data hasil ekstraksi fitur-fitur audio, terlebih dahulu diubah ke dalam representasi biner dengan menggunakan fungsi yang telah diimplementasikan sebelumnya yaitu fungsi float_to_binary. Setelah diubah ke dalam reprentasi binernya, barulah dilakukan proses penyisipan bit-bit paritas serta perubahan data bit sebagai sampel data yang *error* yang kemudian akan diidentifikasi dengan algoritma *hamming code*. Jika memang ternyata terdapat perubahan pada bit data, akan dilakukan proses koreksi, yaitu pengembalian nilai bit data kembali ke keadaan semula.

Setelah melalui proses koreksi, file audio *original* akan dibandingkan dengan audio yang memiliki kekeliruan 1 bit dan audio yang telah dikoreksi dengan *Hamming Code*. Termyata setelah dibandingkan, suara yang dihasilkan pada audio yang memiliki kekeliruan 1 bit tidak sejernih audio *original*. Pada audio yang memiliki kekeliruan bit, sempat terdengan kebisingan (*noise*) pada audionya, padahal pada audio *original*, *noise* tersebut tidaklah ada. Pada audio yang telah melalui koreksi *Hamming Code*, kebisingan yang ada sebelumnya hampir tak terdengar lagi. Hal ini menunjukkan bahwa *Hamming Code* telah terbukti dapat melakukan koreksi pada data yang mengalami prerubahan representasi biner.

IV. KESIMPULAN

Perubahan representasi biner pada suatu data dapat membawa dampak yang besar dan signifikan walaupun data yang berubah hanya 1 bit saja. Hal ini dapat terlihat dari file audio yang memiliki kekeliruan 1 bit ternyata menghasilkan suara yang mengandung *noise*, berbeda dengan *file original* yang suaranya jernih, tanpa adanya *noise*. Dengan adanya algoritma *Hamming Code*, perubahan representasi biner pada data dapat

diidentifikasi. *Hamming Code* juga memungkinkan kita untuk mengetahui letak kesalahan bit data sehingga kita dapat melakukan perubahan pada bit data yang salah agar dapat dihasilkan data sebenarnya yang kita inginkan.

V. UCAPAN TERIMA KASIH

Pertama-tama, penulis ingin megucapkan puji dan syukur kepada Tuhan Yang Maha Esa karena atas berkat rahmatnyalah penulis dapat menyelesaikan makalah ini dengan baik dan tepat waktu. Penulis juga ingin mengucapkan terima kasih kepada Ibu Dr. Fariska Zakhralativa Ruskanda, S.T., M.T., selaku dosen pengajar mata kuliah Matematika Diskrit yang telah membimbing dan memberikan ilmu selama ini kepada penulis sehingga penulis dapat menulis makalah ini dengan baik.

Tak lupa juga, penulis mengucapkan terima kasih kepada teman-teman yang telah banyak memberikan motivasi dan berbagi ilmu selama pembuatan makalah ini berlangsung.

REFERENSI

[1] https://www.andrew.cmu.edu/user/nbier/15110/lectures/lec15a_sound_video.pdf diakses tanggal 11 Desember 2022

[2] https://www.bbc.co.uk/bitesize/guides/zfspfcw/revision/9 diakses tanggal 11 Desember 2022

[3] https://www.geeksforgeeks.org/hamming-code-in-computer-network/ diakses tanggal 11 Desember 2022

[4] Munir, Rinaldi. 2020. Aljabar Boolean (Bagian 1). Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Aljabar-Boolean-(2020)-bagian1.pdf

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Desember 2022

Kandida Edgina Gunawan (13521155)