Otomasi Kode Hamming dengan Menggunakan Aljabar Boolean

Rexy Gamaliel Rumahorbo – 13519010
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13519010@std.stei.itb.ac.id

Di balik kemudahan dalam pengiriman informasi digital yang dapat kita rasakan saat ini, terdapat teknologi yang memampukan hal tersebut terjadi. Algoritma dipakai dalam proses enkripsi dan dekripsi setiap informasi yang dikirim dari satu perangkat ke perangkat lain agar dapat dipulihkan meskipun telah mengalami kerusakan dalam proses pengirimannya. Kode Hamming merupakan salah satu algoritma tersebut yang dapat mendeteksi dan memperbaiki galat 1 bit pada informasi. Dengan menerapkan aljabar Boolean, proses ini dapat diotomasi dengan menggunakan ekspresi Boolean dan rangkaian logika.

***Kata kunci*—Aljabar Boolean, Dekripsi, Enkripsi, Kode Hamming.**

I. PENDAHULUAN

A. Kode Hamming

Saat ini kita dapat menikmati kemudahan dalam memperoleh informasi dengan cepat, aman, dan tanpa repot berkat teknologi. Namun, di balik kemudahan yang kita rasakan ada banyak hal yang memampukan kita merasakan manfaat dari perolehan informasi yang cepat, aman, dan mudah. Mungkin sering terlewatkan dari benak kita bagaimana kita dapat dengan mudahnya memperoleh informasi digital. Setiap teks artikel berita yang kita baca, gambar yang memenuhi lini maya media sosial, video yang menghibur di kala jenuh, maupun sesi kuliah daring menggunakan *online meeting* pada dasarnya hanyalah kumpulan informasi digital dalam representasi 0 dan 1 pada mesin yang kita sebut dengan *handphone*, laptop, televisi, dan lain-lain.

Layaknya media informasi fisik seperti tulisan yang dapat pudar dan rusak, informasi digital juga dapat rusak, baik dalam proses penyimpanan maupun pengiriman. Contohnya, suhu tinggi dan kerusakan fisik pada permukaan CD, dan inteferensi gelombang elektromagnetik pada pengiriman data. Deretan bit 0 dan 1 yang salah, bahkan jika hanya merupakan kesalahan pada satu bit saja, dapat berakibat fatal bagi keseluruhan data. Oleh karena itu diperlukan suatu teknologi untuk memastikan data yang terkirim dapat dipulihkan kembali meskipun terjadi kerusakan.

Kode Hamming merupakan salah satu metode pengkodean yang mampu mengkoreksi kesalahan satu bit pada suatu data. Teknik ini ditemukan oleh Richard W. Hamming pada tahun 1950 untuk membantunya dalam mengotomasi pendeteksian kesalahan dalam penyimpanan bit dalam wujud *punch card*, media yang digunakan untuk menyimpan bit kala itu. Ide dari teknik ini adalah dengan membagi suatu data menjadi beberapa paket dengan panjang bit tertentu, mengekodekannya dengan menambahkan beberapa bit menurut teknik Hamming sebelum kode dikirim, lalu membaca informasi yang diterima dan mendeteksi kesalahannya. Teknik ini terbilang efisien karena membutuhkan tambahan bit yang tidak terlalu banyak untuk pengkodean.

B. Aljabar Boolean

Dalam matematika dan ilmu komputer, aljabar Boolean atau logika Boolean merupakan cabang aljabar dengan dua nilai, yang biasanya dinotasikan dengan *true* dan *false* atau 0 dan 1. Aljabar Boolean ditemukan oleah George Boole pada tahun 1854 melalui tulisannya dalam "*An Investigation of The Laws of Thought*". Dalam bidang komputer, aljabar Boolean memiliki banyak sekali pemanfaatan, salah satunya adalah perancangan gerbang logika dalam arsitektur komputer.

Dalam makalah kali ini akan disajikan otomasi pengkodean enkripisi dan dekripsi Teknik Hamming dengan menerapkan aljabar Boolean.

II. LANDASAN TEORI

A. Aljabar Boolean

1) Definisi Aljabar Boolean

Misalkan *B* adalah sebuah himpunan yang didefinisikan pada dua operator biner, + dan ., dan sebuah operartor uner, ‘, serta 0 dan 1 merupakan dua elemen yang berbeda dari *B*, maka tupel

<*B*, +, ., ‘, 0, 1>

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

a. Identitas
  (i) a + 0 = a
  (ii) a . 1 = a
b. Komutatif
  (i) a + b = b + a
  (ii) a . b = b. a
c. Distributif
  (i) a . (b + c) = (a . b) + (a . c)
  (ii) a + (b . c) = (a + b) . (a + c)
d. Komplemen

Untuk setiap a ∈ *B*, terdapat elemen unik a' ∈ *B* sehingga

(i) a + a' = 1

(ii) a . a' = 0

Karena nilai elemen-elemen *B* tidak didefinisikan, maka ada banyak hal yang termasuk aljabar Boolean. Salah satu contoh aljabar Boolean adalah aljabar logika proposisi dengan tupel <*B*, ∨, ∧, ~, F, T>.

2) Aljabar Boolean 2-Nilai

Aljabar Boolean 2-Nilai merupakan aljabar Boolean yang paling banyak digunakan karena aplikasinya yang luas. Aljabar Boolean 2-Nilai didefinisikan dengan:

(i) B = {0, 1}

(ii) Operator biner: **+** dan **.**, dan operator uner: **'**

(iii) Kaidah untuk operator biner dan operator uner:

a	b	a.b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

a	a'
0	1
1	0

(iv) Memenuhi keempat aksioma aljabar Boolean

3) Hukum Aljabar Boolean

Pada aljabar Boolean berlaku hukum-hukum:

a. Hukum Identitas
- (i) a + 0 = a
- (ii) a . 1 = a

b. Hukum idempoten
- (i) a + a = a
- (ii) a . a = aa = a

c. Hukum komplemen
- (i) a + a' = 1
- (ii) a . a' = aa' = 0

d. Hukum dominasi
- (i) a . 0 = 0
- (ii) a + 1 = 1

e. Hukum involusi
- (i) (a')' = a

f. Hukum penyerapab/absorbs
- (i) a + ab = a
- (ii) a(a + b) = a

g. Hukum komutatif
- (i) a + b = b + a
- (ii) ab = ba

h. Hukum asosiatif
- (i) a + (b + c) = (a + b) + c
- (ii) a (b c) = (a b) c

i. Hukum distributif
- (i) a + (b c) = (a b) + (a c)
- (ii) a (b + c) = (a b) + (a c)

j. Hukum De Morgan
- (i) (a + b)' = a' b'
- (ii) (ab)' = a' + b'

k. Hukum 0/1
- (i) 0' = 1
- (ii) 1' = 0

B. Fungsi Boolean

1) Definisi fungsi boolean

Sebuah fungsi Boolean f didefinisikan dengan

$f : \{0, 1\}^k \rightarrow \{0, 1\}$

di mana k adalah bilangan bulat tak negative yang disebut *arity* dari fungsi f. Beberapa contoh fungsi Boolean antara lain:

$f(x) = 1$

$f(x, y) = x'$

$f(x, y, z) = x'y + x'z$

$g(x, y) = (x + y)(x' + y')$

Setiap variabel dan komplemennya dalam suatu fungsi disebut literal. Contoh: fungsi g(*x*, *y*) = *x*' + *xy*' memiliki 3 literal: *x*', *x*, dan *y*'.

2) Bentuk Kanonik

Ekspresi Boolean yang merepresentasikan gunfi Boolean dapat disajikan dalam dua bentuk kanonik: penjumlahan dari hasil kali/*sum of product* (SOP) dan perkalian dari hasil jumlah/*product of sum* (POS). Pada bentuk SOP, fungsi terdiri atas penjumlahan *minterm*, yakni ekspresi Boolean yang mengandung literal yang lengkap dalam bentuk hasil kali. Sementara itu, pada bentuk POS, fungsi terdiri atas perkalian *maxterm*, yakni ekspresi Boolean yang mengandung literal yang lengkap dalam bentuk hasil jumlah.

Contoh: fungsi *f* dan *g* di bawah merupakan fungsi yang sama:

$f(x, y, z) = xyz + x'yz + xyz' + x'yz' + xy'z$

$g(x, y, z) = (x + y' + z')(x' + y' + z')(x' + y' + z')$

di mana *xyz* adalah salah satu *minterm* dari f dan *x' + y' + z'* adalah salah satu *maxterm* dari g.

3) Minterm dan maxterm

Minterm dan *maxterm* pada sebuah fungsi dalam bentuk kanonik dapat disimbolkan dengan m_D dan M_D berturut-turut sesuai konvensi sebagai berikut:

Untuk fungsi *f* dengan *n* buah parameter terurut x_i, $i = n-1, \ldots, 2, 1, 0$, misalkan sebuah bilangan biner *B* dengan *n* bit yang setiap bitnya dilabeli dengan b_i, $i = n-1, \ldots, 2, 1, 0$, secara berurut dari *most-significant bit* ke *least-significant bit* sedemikian sehingga:

(i) untuk *minterm*, untuk setiap parameter x_i, b_i bernilai 0 jika x_i dinyatakan dalam bentuk komplemen; b_i bernilai 1 jika x_i dinyatakan tanpa komplemen,

(ii) untuk *maxterm*, untuk setiap parameter x_i, b_i bernilai 1 jika x_i dinyatakan dalam bentuk komplemen; b_i bernilai 0 jika x_i dinyatakan tanpa komplemen.

Maka, suatu *minterm* dapat dinyatakan dengan m_D dan suatu *maxterm* dapat dinyatakan dengan M_D di mana *D* adalah representasi desimal dari *B*. Contohnya: untuk sebuah fungsi *f* dengan parameter *x* dan *y*, *minterm* *x*'*y*' berkorespondensi dengan $B = 00_2$, $D = 0$, sehingga dapat dinyatakan dengan m_0. Sementara itu, pada fungsi *g* dengan parameter *u*, *v*, dan *w*, sebuah *maxterm* *u*' + *v* + *w*' berkorespondensi dengan $B = 101_2$, $D = 5$, sehingga dapat dinyatakan dengan M_5.

4) Konversi Antar Bentuk Kanonik

Suatu fungsi dalam bentuk kanonik dapat dinyatakan dengan notasi $\sum$ dan $\prod$. Contohnya:

$$\begin{aligned} f(x,y,z) &= (x+y+z)(x+y'+z)(x'+y+z)(x'+y+z') \\ &= M_0 M_2 M_4 M_5 \\ &= \prod(0,2,4,5) \end{aligned}$$

Hubungan antar *minterm* dan *maxterm* adalah $m_i' = M_i$, sehingga fungsi *f* di atas dapat dinyatakan dengan

$$\begin{aligned} f(x,y,z) &= \left(f'(x,y,z)\right)' \\ &= \left(\left(\prod(0,2,4,5)\right)'\right)' \\ &= \left(\prod(1,3,6,7)\right)' \\ &= (M_1 M_3 M_6 M_7)' \\ &= M_1' + M_3' + M_6' + M_7' \\ &= m_1 + m_3 + m_6 + m_7 \\ &= \sum(1,3,6,7) \\ &= \prod(0,2,4,5) \end{aligned}$$

5) Peta Karnaugh

Peta Karnaugh adalah salah satu metode yang digunakan untuk menyederhanakan fungsi Boolean. Suatu fungsi disederhanakan agar didapat bentuk ekivalennya dengan jumlah literal dan operasi yang lebih sedikit. Hal ini dilakukan agar dalam pengaplikasiannya lebih efektif, misal dalam komputasi logika dan perancangan gerbang logika.

Peta Karnaugh merupakan sebuah visualisasi ekspresi Boolean suatu fungsi yang berwujud sebuah tabel yang kolom dan barisnya merepresentasikan setiap literal atau kombinasi literal yang mungkin. Baris dan kolom disusun sedemikian rupa sehingga untuk setiap pasang sel yang bersisian, sel-sel tersebut hanya berbeda satu literal. Metode ini efektif untuk memvisualisasikan fungsi Boolean dengan banyak parameter yang relatif sedikit.

Setiap sel di peta Karnaugh merepresentasikan *minterm*–jika suatu *minterm* terdapat dalam fungsi, maka sel yang merepresentasikan *minterm* tersebut bernilai 1. Penomoran baris dan kolom tabel serupa dengan aturan penomoran pada penomoran *minterm* pada subbagian *minterm* dan *maxterm*, yakni 0 jika literal dinyatakan dalam bentuk komplemen dan 1 jika literal tidak dinyatakan dalam komplemen.

WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Gambar 1 Peta Karnaugh 4 variabel[4]

Penyederhanaan fungsi Boolean dengan peta Karnaugh dilakukan dengan mencari pasangan dobel, kuad, oktet, dst. pada peta Karnaugh. Kumpulan 2^k sel bernilai 1 yang berada pada "daerah" yang sama dapat dikelompokkan menjadi pasangan dobel, kuad, oktet, dst. dimulai dari kumpulan dengan jumlah terbesar. Setiap kumpulan bisa beririsan, namun memiliki sel yang unik dari kumpulan lainnya.

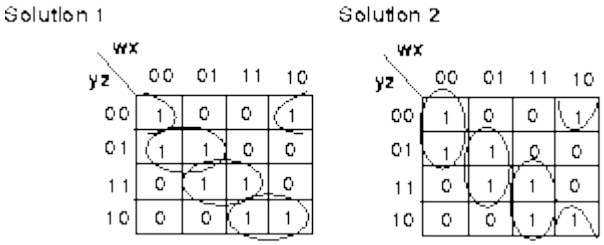
Contohnya, sebuah fungsi *f* berikut dapat dinyatakan dengan peta Karnaugh berikut.

$$f(w,x,y,z) = \sum(0,1,5,7,8,10,14,15)$$

yz \ wx	00	01	11	10
00	1	0	0	1
01	1	1	0	0
11	0	1	1	0
10	0	0	1	1

Gambar 2 Peta Karnaugh fungsi f

Fungsi *f* dapat disederhanakan menurut pengelompokan berikut.



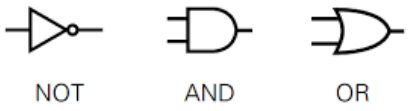
Gambar 3 Variasi solusi peta Karnaugh fungsi f

Sehingga, terdapat 2 bentuk sederhana yang ekivalen dengan *f*, yaitu

$$\begin{aligned} f(w,x,y,z) &= \sum(0,1,5,7,8,10,14,15) \\ &= x'y'z' + w'y'z + xyz + wyz' \quad (solusi\ 1) \\ &= w'x'y' + w'xz + wxy + wx'z' \ (solusi\ 2) \end{aligned}$$

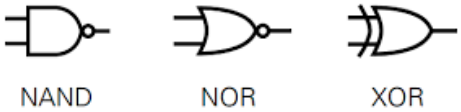
C. Gerbang Logika

Aljabar Boolean dapat diimplementasikan ke dalam rangkaian listrik. Nilai 0 dan 1 masing-masing diwakili oleh arus berhenti dan arus mengalir pada rangkaian. Sementara itu, operator biner + dan . masing-masing diwakili oleh gerbang logika OR dan gerbang logika AND, dan operator uner ' diwakili oleh gerbang logika NOT. Ketiga gerbang logika ini merupakan gerbang logika dasar yang membangun seluruh komposisi yang lebih kompleks.



Gambar 4 Gerbang logika

Beberapa gerbang logika lain dibangun dari ketiga gerbang logika dasar yang telah disebutkan sebelumnya. Gerbang logika lain yang dapat dibentuk dari gerbang NOT, AND, dan OR adalah XOR, XNOR, NOR, dan NAND.



Gambar 5 Gerbang logika lainnya

Gerband NAND dan NOR merupakan negasi dari AND dan

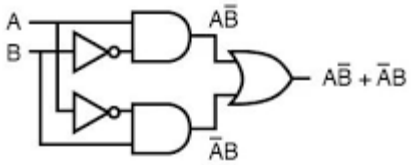
OR, yakni NAND(x, y) = NOT(AND(x, y)) dan NOR(x, y) = NOT(OR(x, y)). Sementara itu, gerbang XOR (biasa disimbolkan $\oplus$ dalam ekspresi Boolean) memenuhi tabel kebenaran

a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0

Operasi $\oplus$ pada ekspresi Boolean dapat dinyatakan dengan kombinasi + dan . yakni

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

atau dalam bentuk gerbang logika



D. Kode Hamming

1) Error-correcting code

Error-correcting codes (ECC) merupakan barisan kode yang dihasilkan oleh algoritma tertentu untuk mendeteksi dan memperbaiki kesalahan dalam data yang dikirimkan melalui kanal yang berpotensi merusak data[2]. Ide utama dari adalah pengirim data mengenkripsikan pesannya dengan informasi tambahan/*redundant* dalam bentuk ECC. Informasi tambahan ini memampukan penerima untuk mendeteksi galat/*error* dalam ukuran yang terbatas. Suatu algoritma ECC semakin efektif dan efisien jika mampu mendeteksi dan/atau memperbaiki sebanyak mungkin galat dengan sesedikit mungkin informasi tambahan. *Error-correcting code* pertama yang diciptakan adalah kode Hamming(7,4), ditemukan oleah Richard W. Hamming pada tahun 1950.

Block code atau kode blok merupakan salah satu jenis ECC. Kode blok memproses data (disebut juga dengan "pesan" atau "*message*") dalam kumpulan paket dengan panjang bit yang sudah ditentukan dan mengenkripsikannya ke dalam kode blok/*codeword* yang kemudian siap dikirim. Pada sisi penerima data, algoritma ECC diharapkan dapat menentukan letak kesalahan pada data dan memperbaikinya. Kode Hamming merupakan salah satu jenis kode blok. Beberapa kode Hamming paling sederhana yaitu kode Hamming(7,4) dan kode Hamming(15,11). Kedua angka dalam tupel mendefinisikan panjang bit kode blok/*codeword* dan panjang bit pesan/*message*. Misalnya, pada kode Hamming (7,4), pengirim akan membagi data yang ingin dikirim ke dalam blok-blok 4 bit, serta menambahkan 3 bit informasi tambahan, sehingga total bit pada suatu blok pada saat pengiriman adalah 7 bit.

Kode Hamming merupakan salah satu algoritma ECC pertama sehingga memiliki beberapa keterbatasan, yaitu hanya mampu mendeteksi dan memperbaiki kesalahan 1 bit pada data. Sementara itu algoritma *extended Hamming* Code mampu mendeteksi kesalahan 2 bit. Namun, selain dari kekurangan tersebut, algoritma ini terbilang cukup efisien karena semakin besar ukuran blok kode yang digunakan, semakin kecil rasio *redundant bits*, yakni sebesar $\log_2(n+1)$ untuk kode blok berukuran n. Pada makalah ini hanya dibahas kasus di mana maksimal galat yang terjadi adalah 1 bit. Untuk mengatasi galat lebih dari itu dapat digunakan algoritma lain.

2) Enkripsi Kode Hamming

Secara garis besar, proses enkripsi kode Hamming dapat dibagi menjadi 3 tahapan: menghitung jumlah informasi tambahan/*redundant bits*, menentukan posisi informasi tambahan, dan menentukan nilai setiap bit dari informasi tambahan.

Langkah pertama mudah dilakukan, tergantung kode Hamming apa yang ingin dipakai. Jumlah bit informasi tambahan/*redundant bits* dapat langsung ditentukan. Misal pada kode Hamming(15,11) terdapat 15-11 = 4 *redundant bits*.

Pada langkah kedua, setiap informasi tambahan diletakkan pada posisi 2^k, yaitu 1, 2, 4, 8, 16, dst. Pada kode Hamming(15,11) berarti *redundant bits* menempati posisi ke-1,2,4,8 pada kode blok/*codewords*, sementara bit pesan/*message* menempati posisi lainnya secara berurutan, yakni posisi ke-3,5,6,…,15.

Langkah selanjutnya, setiap bit informasi tambahan ditentukan berdasarkan paritas/*parity* dari bit-bit pada pesan. Paritas yang digunakan pada kode Hamming umumnya adalah paritas genap. *Redundant bit* r_i adalah paritas untuk setiap bit pada *codeword* digit ke-j (c_j) di mana posisi bit ke-i dari representasi biner j adalah 1, kecuali *redundant bit* itu sendiri. Contohnya, r_1 adalah paritas genap untuk bit pada posisi 3($1\mathbf{1}_2$), 5($10\mathbf{1}_2$), 7($11\mathbf{1}_2$), 9($100\mathbf{1}_2$), dst., r_2 adalah paritas genap untuk bit pada posisi 3($\mathbf{1}1_2$), 6($1\mathbf{1}0_2$), 7($1\mathbf{1}1_2$), 10($10\mathbf{1}0_2$), 11($10\mathbf{1}1_2$), dst.

Misalkan kita ingin mengenkripsi pesan 10101000110 menurut kode Hamming (15,11). Maka akan terdapat 15-11 = 4 *redundant bits* yang masing-masing akan menempati posisi 1, 2, 4, dan 8. Misalkan bit codeword dinotasikan dengan c_i dan bit pada pesan/*message* awal dinotasikan dengan m_i. Berikut adalah penempatan bit-bit pada kode blok dengan visualisasi tabel dan matriks.

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
r_1	r_2	m_1	r_3	m_2	m_3	m_4	r_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}
		1		0	1	0		1	1	0	0	1	1	0

			1
	0	1	0
	1	1	0
0	1	1	0

Tabel 1 Penempatan bit message dalam representasi kode blok Hamming (15,11)

Nilai *redundant bits* r_i dapat ditentukan sebagai berikut:

(i) $r_1 = 1$ karena $c_3, c_5, c_7, c_9, c_{11}, c_{13}, c_{15}$ memiliki bit 1 sejumlah ganjil yaitu 3 (sehingga total bit 1 sekarang menjadi genap)

(ii) $r_2 = 0$ karena $c_3, c_6, c_7, c_{10}, c_{11}, c_{14}, c_{15}$ memiliki bit 1 sejumlah genap yaitu 4 (sehingga total bit 1 sekarang tetap genap)

(iii) $r_3 = 1$ dan $r_4 = 0$ dengan cara yang sama

Sehingga pada akhirnya didapat kode blok lengkap

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
r_1	r_2	m_1	r_3	m_2	m_3	m_4	r_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}
1	0	1	1	0	1	0	0	1	1	0	0	1	1	0

	1	0	1
1	0	1	0
0	1	1	0
0	1	1	0

Tabel 2 Penempatan bit message dalam representasi kode blok Hamming (15,11)

3) Dekripsi Kode Hamming

Secara garis besar, proses dekripsi kode Hamming terbagi ke dalam 4 tahapan: menentukan jumlah *redundant bits*, menentukan posisi *redundant bits*, mengecek paritas/*parity checking*, mendeteksi dan memperbaiki galat.

Kedua langkah awal serupa dengan tahapan pada enkripsi. Untuk langkah ketiga, dilakukan pengecekan untuk setiap kelompok bit. Langkah ini pada dasarnya sama dengan Langkah ketiga pada proses enkripsi.

Misalkan digunakan kembali contoh kode blok pada bagian sebelumnya, namun penerima/*receiver* menerima kode dengan galat 1 bit pada bit ke-12 (ingat kembali bahwa kode Hamming hanya dapat mendeteksi kesalahan 1 bit) sebagai berikut.

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
r_1	r_2	m_1	r_3	m_2	m_3	m_4	r_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}
1	0	1	1	0	1	0	0	1	1	0	1	1	1	0

	1	0	1
1	0	1	0
0	1	1	0
1	1	1	0

Tabel 3 Penempatan bit message dalam representasi kode blok Hamming (15,11)

Penerima melakukan *parity checking* sebagai berikut:

(i) $c_1, c_3, c_5, c_7, c_9, c_{11}, c_{13}, c_{15}$ memiliki bit 1 sejumlah ganjil, maka tidak ada kesalahan di bit-bit ini

(ii) $c_2, c_3, c_6, c_7, c_{10}, c_{11}, c_{14}, c_{15}$ memiliki bit 1 sejumlah genap, maka tidak ada kesalahan di bit-bit ini

(iii) $c_4, c_5, c_6, c_7, c_{11}, c_{13}, c_{14}, c_{15}$ memiliki bit 1 sejumlah ganjil, maka ada kesalahan di antara bit-bit ini

(iv) $c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}$ memiliki bit 1 sejumlah ganjil, maka ada kesalahan di antara bit-bit ini

Dengan eliminasi kemungkinan yang ada, kita dapat mendapat kesimpulan sebagai berikut. Dengan melihat posisi bit pada matriks, dari pernyataan (i) dan (ii) didapat bahwa bit galat mungkin terdapat di kolom pertama. Dari pernyataan (iii) dan (iv) didapat bahwa bit galat terdapat di baris keempat. Dari kedua kalimat di atas dapat disimpulkan bahwa bit galat terdapat di baris ke-4 kolom ke-1, yakni bit *codeword* ke-12.

Cara lain menentukan posisi galat adalah dengan membuat "*flag*" untuk setiap *redundant bits* yang akan menyala jika terdapat galat pada kelompok bit-bit yang berkorespondensi dengan r_i. Jika tidak ada *flag* yang menyala pada akhir *parity checking*, maka dapat dipastikan tidak terdapat galat. Jika terdapat *flag* menyala maka posisi bit galat dapat ditentukan dengan menjumlahkan posisi *redundant bit* yang *flag*-nya menyala. Misal pada contoh di atas, *flag* dari *redundant bit* r_3 dan r_4 akan menyala. Karena kedua bit ini masing-masing menempati posisi c_4 dan c_8 pada *codeword*, maka bit galat terdapat di posisi 4+8 = 12. Hal ini dimungkinkan oleh konfigurasi sedemikian rupa dari (i) penempatan *redundant bit* pada *codeword* dan (ii) posisi bit-bit yang berkorespondensi dengan *redundant bit*.

Langkah terakhit dalam dekripsi kode Hamming, setelah penerima mendeteksi galat pada bit ke-12, maka nilai bit tersebut diganti, dari 1 menjadi 0.

III. PEMBAHASAN

Secara sekilas, kode Hamming secara alamiah sangatlah erat kaitannya dengan aljabar Boolean. Algoritma penentuan *redundant bits* serta pendeteksian dan perbaikan galat dapat dimodelkan ke dalam fungsi Boolean. Pada pembahasan kali ini, kode Hamming yang akan digunakan adalah kode Hamming (15,11).

Sebagaimana telah dibahas sebelumnya, akan terdapat 4 *redundant bits* dan 11 *message bits*. Keempat *redundant bits* menempati bit ke-1, 2, 4, dan 8 pada blok kode/*codeword*. Perhatikan juga kesebelas bit pesan/*message bits* m_i secara berturut-turut menempati posisi 3, 5, 6, 7, 9, 11, 12, 13, 14, dan 15 pada kode blok/*codeword.*

A. Fungsi Boolean Enkripsi Kode Hamming

Misalkan empat fungsi e_1, e_2, e_3, e_4 adalah fungsi yang menghasilkan *redundant bits* dari 11 parameter *message bits*. Perhatikan bahwa paritas genap dari sekelompok bit adalah sama dengan mengoperasikan operasi logika xor ($\oplus$) untuk setiap bit sehingga menghasilkan 0. Dengan kata lain, suatu *redundant bit* itu sendiri merupakan hasil xor dari setiap *message bits* yang berkorespondensi dengannya.

$$\begin{aligned} r_1 &= e_1(m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}, m_{11}) \\ &= m_1 \oplus m_2 \oplus m_4 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11} \\ &= e_1(c_3, c_5, c_6, c_7, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}) \\ &= c_3 \oplus c_5 \oplus c_7 \oplus c_9 \oplus c_{11} \oplus c_{13} \oplus c_{15} \end{aligned}$$

$$r_2 = e_2(m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}, m_{11})$$
$$= m_1 \oplus m_3 \oplus m_4 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11}$$
$$= e_1(c_3, c_5, c_6, c_7, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15})$$
$$= c_3 \oplus c_6 \oplus c_7 \oplus c_{10} \oplus c_{11} \oplus c_{14} \oplus c_{15}$$

$$r_3 = e_3(m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}, m_{11})$$
$$= m_2 \oplus m_3 \oplus m_4 \oplus m_8 \oplus m_9 \oplus m_{10} \oplus m_{11}$$
$$= e_1(c_3, c_5, c_6, c_7, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15})$$
$$= c_5 \oplus c_6 \oplus c_7 \oplus c_{12} \oplus c_{13} \oplus c_{14} \oplus c_{15}$$

$$r_4 = e_4(m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}, m_{11})$$
$$= m_5 \oplus m_6 \oplus m_7 \oplus m_8 \oplus m_9 \oplus m_{10} \oplus m_{11}$$
$$= e_1(c_3, c_5, c_6, c_7, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15})$$
$$= c_9 \oplus c_{10} \oplus c_{11} \oplus c_{12} \oplus c_{13} \oplus c_{14} \oplus c_{15}$$

B. Fungsi Boolean Enkripsi Kode Hamming

Misalkan empat fungsi d_1, d_2, d_3, d_4 merupakan *flag g* dari r_1, r_2, r_3, r_4, berturut-turut di mana *flag* menyala jika fungsi *d* bernilai 1.

$$g_1 = d_1(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15})$$
$$= (c_1 \oplus c_3 \oplus c_5 \oplus c_7 \oplus c_9 \oplus c_{11} \oplus c_{13} \oplus c_{15})'$$

$$g_2 = d_2(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15})$$
$$= (c_2 \oplus c_3 \oplus c_6 \oplus c_7 \oplus c_{10} \oplus c_{11} \oplus c_{14} \oplus c_{15})'$$

$$g_3 = d_1(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15})$$
$$= (c_4 \oplus c_5 \oplus c_6 \oplus c_7 \oplus c_{12} \oplus c_{13} \oplus c_{14} \oplus c_{15})'$$

$$g_4 = d_1(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15})$$
$$= (c_8 \oplus c_9 \oplus c_{10} \oplus c_{11} \oplus c_{12} \oplus c_{13} \oplus c_{14} \oplus c_{15})'$$

Berdasarkan *flag* menyala, nilai setiap bit pada *codeword* dimanipulasi/diperbaiki. Namun karena bit yang penting adalah *message bits*, maka manipulasi cukup dilakukan pada bit-bit tersebut. Manipulasi dilakukan berdasarkan status dari *flag* yang menyala serta nilai *message bit* yang diterima, berdasarkan hubungan ekivalensi/"jika dan hanya jika". Misalkan *message bit* yang telah dimanipulasi dinotasikan dengan p_i.

$$p_1 = (g_1) \leftrightarrow m_1'$$
$$= (g_1 + m_1)(g_1' + m_1')$$
$$= (g_1 g_2' g_3' g_4') \oplus m_1'$$

$$p_2 = (g_1' g_2 g_3' g_4') \leftrightarrow m_2'$$
$$= (g_1' g_2 g_3' g_4') \oplus m_2'$$

$$p_3 = (g_1 g_2 g_3' g_4') \leftrightarrow m_3'$$
$$= (g_1 g_2 g_3' g_4') \oplus m_3'$$

$$p_4 = (g_1' g_2' g_3 g_4') \leftrightarrow m_4'$$
$$= (g_1' g_2' g_3 g_4') \oplus m_4'$$

$$p_5 = (g_1 g_2' g_3 g_4') \leftrightarrow m_5'$$
$$= (g_1 g_2' g_3 g_4') \oplus m_5'$$

$$p_6 = (g_1' g_2 g_3 g_4') \leftrightarrow m_6'$$
$$= (g_1' g_2 g_3 g_4') \oplus m_6'$$

$$p_7 = (g_1 g_2 g_3 g_4') \leftrightarrow m_7'$$
$$= (g_1 g_2 g_3 g_4') \oplus m_7'$$

$$p_8 = (g_1' g_2' g_3' g_4) \leftrightarrow m_8'$$
$$= (g_1' g_2' g_3' g_4) \oplus m_8'$$

$$p_9 = (g_1 g_2' g_3' g_4) \leftrightarrow m_9'$$
$$= (g_1 g_2' g_3' g_4) \oplus m_9'$$

$$p_{10} = (g_1' g_2 g_3' g_4) \leftrightarrow m_{10}'$$
$$= (g_1' g_2 g_3' g_4) \oplus m_{10}'$$

$$p_{11} = (g_1 g_2 g_3{}' g_4) \leftrightarrow m_{11}'$$
$$= (g_1 g_2 g_3{}' g_4) \oplus m_{11}'$$

IV. KESIMPULAN

Kode Hamming merupakan salah satu algoritma *error-corection code*/ECC yang mampu mendeteksi dan memperbaiki galat 1 bit pada suatu informasi dengan panjang tertentu. Enkripsi dan dekripsi kode Hamming menggunakan *parity checking* berdasarkan bit pesan yang telah diposisikan sedemikian rupa pada blok kode, dan hasilnya disimpan pada bit informasi tambahan/*redundant bits*. Proses enkripsi dan dekripsi kode Heming ini dapat diotomasi dengan merancang fungsi Boolean yang sesuai.

V. UCAPAN TERIMA KASIH

Terima kasih saya ucapkan sebesar-besarnya kepada Tuhan Yang Maha Esa karena penyusunan makalah ini dapat terlaksanan dengan baik. Terima kasih juga kepada para dosen mata kuliah IF2120 yang telah membimbing dan memberi kesempatan dalam penyusunan makalah ini. Saya menyadari masih banyak kekurangan dalam penulisan makalah ini. Semoga informasi yang disajikan dapat bermanfaat bagi para pembaca.

REFERENCES

[1] George Boole (1848). "The Calculus of Logic," Cambridge and Dublin Mathematical Journal III: 183–98.W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[2] Glover, Neal; Dudley, Trent (1990). Practical Error Correction Design For Engineers (Revision 1.1, 2nd ed.). CO, USA: Cirrus Logic. ISBN 0-927239-00-0. ISBN 978-0-927239-00-4.B. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished.

[3] https://www.encyclopediaofmath.org/index.php/Boolean_function (diakses Jumat, 11-12-2020)

[4] https://www.tutorialspoint.com/digital_circuits/digital_circuits_k_map_method.htm (diakses Jumat, 11-12-2020)

[5] https://www.tutorialspoint.com/error-correcting-codes-hamming-codes (diakses Jumat, 11-12-2020)

[6] Munir, Rinaldi. 2020. Aljabar Boolean (Bag. 1) Bahan Kuliah IF 2120 Matematika Diskrit. Bandung. Prodi Informatika, Sekolah Teknik Elektri dan Informatika, (diakses pada 11 Desember 2020)

[7] Munir, Rinaldi. 2020. Aljabar Boolean (Bag. 2) Bahan Kuliah IF 2120 Matematika Diskrit. Bandung. Prodi Informatika, Sekolah Teknik Elektri dan Informatika, (diakses pada 11 Desember 2020)

[8] Munir, Rinaldi. 2020. Aljabar Boolean (Bag. 3) Bahan Kuliah IF 2120 Matematika Diskrit. Bandung. Prodi Informatika, Sekolah Teknik Elektri dan Informatika, (diakses pada 11 Desember 2020)

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, 11 Desember 2020

Rexy Gamaliel Rumahorbo - 13519010