KOMPUTASI ARITMATIK GF(2^m) UNTUK KONSTRUKSI KODE *BOSE-CHAUDHURI-HOCQUENGHEM* (BCH)

MEI HARIYANTO

DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT PERTANIAN BOGOR
2004

ABSTRAK

MEI HARIYANTO. Komputasi Aritmatik GF(2^m) Untuk Konstruksi Kode *Bose-Chaudhuri-Hocquenghem* (BCH). Dibimbing oleh SUGI GURITMAN dan FIRMAN ARDIANSYAH.

Kualitas informasi dalam komunikasi data digital adalah masalah penting. Apabila informasi ditransfer melalui saluran terganggu maka kemungkinan besar akan terjadi galat. Akibatnya, ada permintaan pengiriman ulang. Apabila informasi yang ditransfer ulang tersebut tetap melalui saluran terganggu maka yang muncul adalah galat-galat berikutnya. Akibatnya terjadi permintaan pengiriman ulang berikutnya. Tentunya hal ini sangat merepotkan dan memerlukan banyak waktu.

Semua informasi akan diubah menjadi katakode oleh perangkat *encoder*. *Encoder* akan menambahkan bit pada informasi. Kode merupakan himpunan yang anggotanya katakode. Kode diciptakan sedemikian sehingga *decoder* dapat mendeteksi dan mengoreksi galat. Dengan demikian, kode mampu memperbaiki kualitas informasi yang ditransfer. Untuk menciptakan kode diperlukan operasi aritmatik GF(2^m), termasuk kode BCH yang merupakan keluarga kode siklik.

Penelitian ini bertujuan untuk mempelajari dan membuat algoritma proses konstruksi kode BCH menggunakan komputasi aritmatik GF(2^m) sehingga dapat diimplementasikan dalam bentuk program. Dalam penelitian ini penulis menyusun algoritma dan implementasi konstruksi GF(2^m), konstruksi kode BCH, dan proses *encoding* kode BCH.

GF(2^m) memiliki peran yang sangat penting dalam konstruksi kode BCH. GF(2^m) harus diciptakan terlebih dulu karena operasi-operasinya dibutuhkan untuk proses pembentukan polinomial minimal. Untuk menciptakan GF(2^m) diperlukan sebuah polinomial primitif dengan koefisien-koefisien atas GF(2).

Setiap koset siklotomik memiliki polinomial minimal yang unik. Dengan memanfaatkan proses perkalian dan penjumlahan GF(2^m), pangkat α pembuat siklik, dan setiap elemen koset maka dapat dibentuk polinomial minimal.

Setiap polinomial generator $g(x)$ memiliki *designed distance* tertentu, menentukan besarnya dimensi kode, menentukan besarnya *actual distance*, menentukan bentuk sirkuit *encoding*, dan menentukan banyaknya bit simbol cek yang ditambahkan pada simbol pesan.

KOMPUTASI ARITMATIK GF(2^m) UNTUK KONSTRUKSI KODE *BOSE-CHAUDHURI-HOCQUENGHEM* (BCH)

MEI HARIYANTO

Skripsi
Sebagai salah satu syarat untuk memperoleh gelar
Sarjana Komputer
pada
Program Studi Ilmu Komputer

DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT PERTANIAN BOGOR
2004

Judul Skripsi : Komputasi Aritmatik GF(2^m) Untuk Konstruksi Kode *Bose-Chaudhuri-Hocquenghem* (BCH)
Nama : Mei Hariyanto
NRP : G06499004
Departemen : Ilmu Komputer

Menyetujui,

Dr. Ir. Sugi Guritman
Pembimbing I

Firman Ardiansyah, S. Kom.
Pembimbing II

Mengetahui,

Ir. Julio Adisantoso, M. Komp.
Ketua Program Studi

Ir. Agus Buono, M. Si, M. Komp.
Ketua Departemen

RIWAYAT HIDUP

Penulis dilahirkan di Cilacap pada tanggal 11 Mei 1980 dari orang tua yang bernama Tijo Noto Raharjo dan Karsem. Penulis merupakan anak pertama dari dua bersaudara.

Tahun 1999 penulis lulus dari SMU Negeri 1 Cilacap dan pada tahun yang sama lulus seleksi masuk IPB melalui jalur Undangan Seleksi Masuk IPB. Penulis memilih Program Studi Ilmu Komputer, Departemen Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam.

Selama mengikuti perkuliahan, penulis pernah melakukan Praktik Kerja Lapangan di Pusat Penelitian Biologi-LIPI Bogor pada bulan Januari sampai dengan bulan Maret 2003.

KATA PENGANTAR

Puji dan syukur penulis panjatkan ke hadirat Allah SWT atas segala rahmat, petunjuk dan karunia-Nya sehingga tugas akhir ini dapat diselesaikan. Tema yang dipilih dalam penelitian ini adalah komputasi aritmatik GF(2^m) untuk konstruksi kode *Bose-Chaudhuri-Hocquenghem* (BCH).

Terima kasih yang sebesar-besarnya penulis ucapkan kepada Bapak Dr. Ir. Sugi Guritman selaku pembimbing I dan Bapak Firman Ardiansyah, S.Kom selaku pembimbing II yang telah memberikan saran dan bimbingan selama pengerjaan tugas akhir ini. Selanjutnya, penulis juga mengucapkan terima kasih kepada:

1. Bapak, Ibu, dan Adikku atas do'a, dukungan, dan kasih sayang yang selalu diberikan.
2. Ndhukku sayang, Dian, yang selalu memberikan dukungan dan masukan yang positif.
3. Danny, Teguh, Baskoro, Siska atas semua bantuannya membantu penulis. Semua rekan *Ilkomerz*'36 atas dukungan yang diberikan.
4. Tosan, Andre, Heru atas semangat, canda tawa, dan kebersamaannya.
5. Komputerku yang telah setia menemani penulis selama pengerjaan tugas akhir.
6. Semua rekan *Ilkomerz* atas segala dukungan dan kebersamaannya.
7. Seluruh staf pengajar dan pegawai Departemen Ilmu Komputer atas bantuannya.
8. Semua pihak yang tidak dapat disebutkan satu persatu.

Semoga tugas akhir ini dapat bermanfaat, dan untuk semua pihak yang terlibat dalam penyusunan tugas akhir ini, semoga mendapat balasan yang setimpal.

Bogor, Agutus 2004

Mei Hariyanto

DAFTAR ISI

Halaman

DAFTAR TABEL

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR LAMPIRAN

Halaman

PENDAHULUAN

Latar Belakang

Berbicara masalah komunikasi data digital dari suatu media ke media lain melalui saluran (*channel*), tidak terlepas dari masalah keamanan dan kualitas informasi. Keamanan sangat dibutuhkan untuk melindungi informasi dari pihak-pihak yang tidak diinginkan. Hal ini terjadi apabila saluran yang digunakan diasumsikan tidak aman.

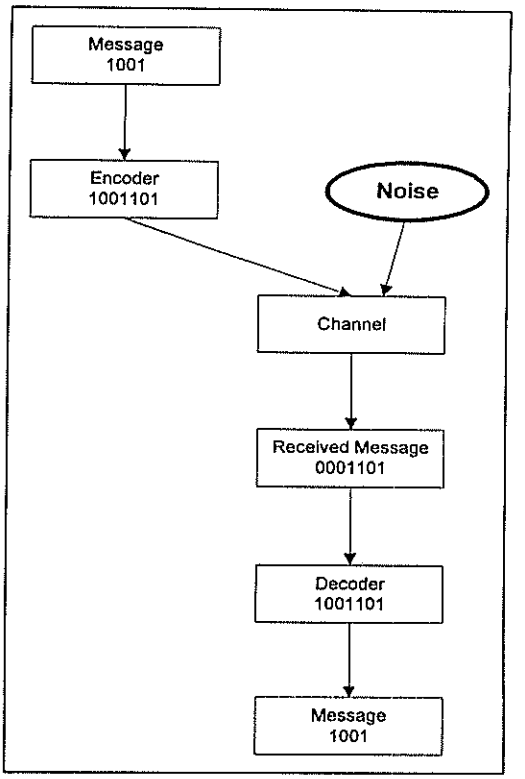
Yang tidak kalah penting dalam komunikasi data digital adalah masalah kualitas informasi. Apabila informasi ditransfer melalui saluran terganggu (*noisy channel*) maka kemungkinan besar akan terjadi galat (*error*). Akibatnya ada permintaan pengiriman ulang. Apabila informasi yang ditransfer ulang tetap melalui saluran terganggu maka yang muncul adalah galat-galat berikutnya. Akibatnya terjadi permintaan pengiriman ulang berikutnya. Tentunya hal ini sangat merepotkan dan memerlukan banyak waktu.

Menurut Pless (1989) saluran komunikasi dapat berupa: saluran telepon, jaringan komputer, jaringan radio dengan frekuensi tinggi atau jaringan komunikasi satelit. Gangguan bisa berasal dari: manusia, kerusakan alat, interferensi, petir, atau goresan pada *disk*.

Informasi dalam komunikasi data digital direpresentasikan dalam bentuk blok atau barisan simbol 0 dan 1 yang dikenal dengan *bitstring*. Galat yang dimaksudkan disini adalah terjadinya perubahan simbol 0 menjadi simbol 1 atau sebaliknya perubahan simbol 1 menjadi 0. Pless (1989) menegaskan bahwa mendeteksi terjadinya galat lebih mudah daripada mengoreksinya.

Gambar 1 menunjukan bahwa sebelum dikirimkan, semua informasi akan diubah menjadi katakode (*codeword*) oleh perangkat *encoder*. *Encoder* akan menambahkan bit pada informasi, proses ini disebut *encoding*. Kode (*code*) merupakan himpunan yang anggotanya katakode. Kode diciptakan sedemikian sehingga *decoder* dapat mendeteksi dan mengoreksi galat, proses ini disebut *decoding*. Dengan demikian kode mampu memperbaiki kualitas informasi yang ditransfer.

Untuk menciptakan kode diperlukan operasi aritmatik GF(2^m). Demikian juga untuk menciptakan kode BCH yang merupakan keluarga kode siklik harus diciptakan GF(2^m) terlebih dahulu.



Gambar 1. Saluran komunikasi terganggu menurut Pless (1989).

Tujuan

Penelitian ini bertujuan untuk mempelajari dan membuat algoritma proses konstruksi kode BCH menggunakan komputasi aritmatik GF(2^m) sehingga dapat diimplementasikan dalam bentuk program.

Ruang Lingkup

Ruang lingkup penelitian terdiri atas:

1. Menyusun algoritma konstruksi GF(2^m) dan implementasinya.
2. Menyusun algoritma konstruksi kode BCH dan implementasinya.
3. Menyusun algoritma proses *encoding* kode BCH dan implementasinya.

TINJAUAN PUSTAKA

Landasan Teori Untuk Konstruksi GF(2^m)

1. Landasan Teori Aljabar

Group G adalah suatu himpunan dengan operasi * dengan aksioma-aksioma sebagai berikut:

- G bersifat tertutup terhadap operasi *. Jika $g, h \in G$ maka $g * h \in G$.
- Operasi * bersifat asosiatif.
- G memiliki unsur identitas e dengan $e * g = g * e$, untuk setiap $g \in G$.
- Setiap $g \in G$ memiliki invers g^{-1} dengan $g * g^{-1} = g^{-1} * g = e$.

Group G disebut *group* komutatif jika $g * h = h * g$ untuk semua $g, h \in G$.

Contoh *group*: Himpunan bilangan bulat dengan operasi penjumlahan.

Ring R adalah suatu himpunan dengan operasi penjumlahan + dan perkalian • dengan aksioma-aksioma sebagai berikut:

- R adalah *group* komutatif terhadap operasi penjumlahan.
- R bersifat asosiatif terhadap operasi perkalian.
- Bersifat distributif perkalian terhadap penjumlahan, yaitu: $a \bullet (b + c) = (a \bullet b) + (a \bullet c)$, untuk semua $a, b, c \in R$.

Contoh *ring*: Himpunan bilangan bulat dengan operasi penjumlahan.

Field F adalah suatu himpunan dengan operasi penjumlahan + dan perkalian • dengan aksioma-aksioma sebagai berikut:

- F adalah *group* komutatif terhadap operasi penjumlahan.
- $F - \{0\}$ adalah *group* komutatif terhadap operasi perkalian.
- Bersifat distributif perkalian terhadap penjumlahan, yaitu: $a \bullet (b + c) = (a \bullet b) + (a \bullet c)$, untuk semua $a, b, c \in R$.

Contoh *field*: Himpunan bilangan nyata dengan operasi penjumlahan dan perkalian.

Contoh bukan *field*: Himpunan bilangan bulat terhadap operasi perkalian, karena inversnya bukan merupakan bilangan bulat. Misalnya: invers dari 2 adalah ½. Sedangkan ½ sendiri bukan bilangan bulat.

***Field* berhingga** adalah *field* yang memiliki jumlah elemen yang berhingga yang disebut orde (*order*). *Field* berhingga dibuktikan oleh matematikawan Perancis Evariste Galois (1811-1832), sehingga *field* berhingga disebut Galois Field (GF).

Contoh *field* berhingga: *Field* berorde prima p dan operasi aritmatik mod p. Misalkan Z_7 atau $GF(7^1) = \{0, 1, 2, 3, 4, 5, 6\}$. Secara khusus untuk $GF(2^1) = \{0, 1\}$disebut *field* biner. *Field* biner memiliki operasi XOR untuk operasi jumlah mod 2 dan operasi AND untuk operasi kali mod 2.

$GF(2^m)$ merupakan *field* biner yang beranggotakan semua *bitstring* dengan panjang m yang konstruksi dan operasinya didasarkan pada beberapa teori-teori berikut:

- **Teorema 1** *Pasti ada sebuah field dengan orde q jika dan hanya jika q merupakan pangkat prima ($q = p^h$, dengan p prima dan h integer positif).* Bukti: Lihat Hill (1994).

- **Teorema 2** *Untuk p prima dan integer $m \geq 1$, ada sebuah field berorde p^m, dilambangkan dengan $GF(p^m)$.* Bukti: Lihat MacWilliams dan Sloane (1983).

- **Teorema 3** *Misalkan $F = GF(p^m)$ dan F^* adalah elemen taknol dari F. F^* merupakan group perkalian siklik berorde $p^m - 1$.* Bukti: Lihat MacWilliams dan Sloane (1983).

  Suatu *group G* berorde r dengan operasi perkalian disebut siklik jika ada $a \in G$ sehingga $G = \{a^0 = 1, a, a^2, ..., a^{r-1}\} = \langle a \rangle$. Dengan a disebut generator dari G. Jika $\alpha \in GF(p^m)$ adalah generator dari F^* maka α disebut *elemen primitif.*

- ***Lemma* 1** *Dalam field berhingga dengan karakteristik p:*

$$(x + y)^p = x^p + y^p.$$

Bukti: Lihat MacWilliams dan Sloane (1983).

2. Polinomial Minimal

Menurut Hill (1986), jika $f(x) = f_0 + f_1x + f_2x^2 + ... + f_nx^n$ adalah sebuah polinomial dengan $f_n \neq 0$, maka n disebut derajat dari $f(x)$, f_n disebut koefisien pemimpin (*leading coefficient*). Suatu polinomial dikatakan *monic* jika memiliki koefisien pemimpin sama dengan 1. Polinomial *reciprocal* dari $f(x)$ adalah $f_n + f_{n-1}x + ... + f_2x^{n-2} + f_1x^{n-1} + f_0x^n$.

Definisi 1 (MacWilliams & Sloane 1983) *Polinomial minimal atas GF(p) dari β adalah polinomial monic berderajat terkecil M(x) dengan koefisien anggota GF(p) sehingga $M(\beta) = 0$. Dalam hal ini β disebut akar dari M(x).*

Misalkan $M(x)$ adalah polinomial minimal dari $\beta \in GF(p^m)$. Sifat-sifat polinomial minimal $M(x)$ meliputi:

- *Sifat 1. M(x) irreducible.*

  Tidak ada polinomial lain dengan derajat < derajat $M(x)$ yang merupakan faktor dari $M(x)$.

- *Sifat 2. Jika f(x) adalah polinomial atas GF(p) sedemikian sehingga f(β)=0 maka M(x) | f(x).*
- *Sifat 3. $M(x) \mid x^{p^m} - x$.*
- *Sifat 4. Derajat $M(x) \leq m$.*
- *Sifat 5. Polinomial minimal dari elemen primitif $GF(p^m)$ memiliki derajat m. Polinomial ini disebut polinomial primitif.*
- *Sifat 6. β dan β^p memiliki polinomial minimal yang sama.*

Setiap elemen *field* yang memiliki polinomial minimal yang sama disebut *conjugates*.

Definisi 2 (MacWilliams & Sloane 1983) *Operasi perkalian dengan p dan membagi integer-integer dengan mod p^m-1 menjadi himpunan-himpunan disebut koset siklotomik (cyclotomic cosets).*

Koset siklotomik $C_s = \{s, ps, p^2s, p^3s, ..., p^{l-1}s\}$ dimana l adalah bilangan *integer* positif terkecil sehingga $p^l s \equiv s \pmod{p^m - 1}$. Sebagai contoh, koset-koset siklotomik mod 15 dengan $p = 2$ terdiri dari:

$C_0 = \{0\}$,
$C_1 = \{1, 2, 4, 8\}$,
$C_3 = \{3, 6, 12, 9\}$,
$C_5 = \{5, 10\}$,
$C_7 = \{7, 14, 13, 11\}$.

- *Sifat 7. Jika i terdapat dalam C_s maka*

$$M^{(i)}(x) = \prod_{i \in C_s} (x - \alpha^i).$$

Teorema 4 *Misalkan M(x) adalah polinomial minimal dari β elemen $GF(2^m)$, e adalah derajat M(x). Maka e adalah integer terkecil sehingga $\beta^{2^e} = \beta$, $e \leq m$.* Bukti: Lihat Shu dan Costello (1983).

Landasan Teori Untuk Kontruksi Kode BCH

1. Kode Linear

Menurut Guritman (2003), misalkan F_q adalah *field* berhingga berukuran kuasa prima q, dan misalkan $(F_q)^n$ adalah ruang vektor berdimensi n atas F_q (dalam hal ini F_q adalah $GF(2^m)$).

Kode linear dengan panjang n didefinisikan sebagai subruang $C \subseteq (F_q)^n$. Anggota dari suatu kode disebut katakode. Kode linear yang dimaksudkan di sini adalah *kode blok linear*.

Kode linear C memiliki parameter:

a. Jarak minimum *d*.

Definisi 3 (Guritman 2003) *Jarak (Hamming distance) antara dua vektor $x, y \in (F_q)^n$ -- dinotasikan d(x,y) -- adalah banyaknya posisi digit dari x dan y dimana simbol mereka berbeda. Jarak minimum (minimum Hamming distance) dari suatu kode linear C didefinisikan:*

$$d(C) = min \{d(x,y) \mid x,y \in C, x \neq y\}.$$

Definisi 4 (Guritman 2003) *Bobot (Hamming weight) dari suatu vektor $x \in (F_q)^n$, dinotasikan wt(x), adalah banyaknya simbol taknol dalam x. Bobot minimum (minimum Hamming weight) dari suatu kode C didefinisikan:*

$$wt(C) = min \{wt(x) \mid x \in C, x \neq 0\}.$$

Berdasarkan Definisi 3 dan 4 maka diperoleh $d(x, y) = wt(x-y)$.

Teorema 5 *Jarak minimum dari suatu kode linear C adalah bobot minimum dari sembarang katakode taknol.*

Bukti:

$$\begin{aligned} d(C) &= \min \{d(x, y) \mid x,y \in C, x \neq y\} \\ &= \min \{wt(x - y) \mid x, y \in C, x \neq y\} \\ &= \min \{wt(z) \mid z \in C, z \neq 0\} \\ &= wt(C). \text{ (terbukti)} \end{aligned}$$

Teorema 6 *Sebuah dengan jarak minimum d dapat mengoreksi [½(d-1)] galat. Jika d genap, kode akan dapat mengoreksi ½(d-2) dan sekaligus dapat mendeteksi d/2 galat.* Bukti: Lihat MacWilliams dan Sloane (1983).

b. Panjang kode *n*.

Setiap katakode dalam kode linear C akan memiliki panjang tetap n disebut *blok* dan terbagi menjadi dua bagian yaitu: simbol pesan dan simbol cek.

c. Dimensi *k*.

Kode linear C memiliki dimensi dengan ukuran k yang menentukan banyaknya katakode. Menurut MacWilliams dan Sloane (1983) setiap kode akan memiliki katakode sebanyak 2^k. Dimensi k merupakan panjang dari simbol pesan.

Untuk menciptakan kode harus mempertimbangkan parameter-parameter tersebut. Jarak minimum d dibuat sebesar mungkin sebab semakin besar d semakin banyak galat yang bisa dikoreksi. Panjang kode n dibuat sekecil mungkin. Hal ini berkaitan dengan penggunaan memori dalam implementasinya dan waktu proses *encoding* dan *decoding*. Dimensi k dibuat sebesar

mungkin karena semakin besar k semakin banyak pesan yang bisa di*encoding*.

Konstruksi kode berarti sama dengan mengonstruksi matriks paritas **H** atau matriks generator **G**.

Matriks cek paritas H adalah matriks dengan ukuran $(n - k)$ x n dalam bentuk $\mathbf{H} = [\mathbf{A}|\mathbf{I}_{n-k}]$ dengan:

a. **A** merupakan matriks dengan ukuran tetap $(n - k)$ x k.

b. $\mathbf{I}_{n-k}$ merupakan matriks identitas dengan ukuran $(n - k)$ x $(n - k)$ (Mac Williams & Sloane 1983).

Berdasarkan pengertian di atas maka kode didefinisikan sebagai:

$C = \{x \in (F_q)^n \mid \mathbf{H}x^{tr} = 0\}$.

Misalnya $\boldsymbol{u} = (u_1 u_2 \ldots u_k)$ akan diubah menjadi menjadi katakode $x = (x_1 x_2 \ldots x_n)$ dengan $n \geq k$ maka x berbentuk $x_1 = u_1$, $x_2 = u_2$, ..., $x_k = u_k$ diikuti dengan simbol cek $x_{k+1}, \ldots, x_n$.

Contoh:

Misalkan $\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$ dan pesan $\boldsymbol{u} = (11)$, maka:

$$\mathbf{H}x^{tr} = 0,$$

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

sehingga diperoleh sistem persamaan linear:

$1 + x_3 = 0 \Leftrightarrow x_3 = -1 = 1$,

$1 + 1 + x_4 = 0 \Leftrightarrow x_4 = 0$.

Dengan demikian pesan $\boldsymbol{u}$ = (11) memiliki katakode x = (1110).

Matriks generator G adalah matriks dengan ukuran k x n dalam bentuk $\mathbf{G} = [\mathbf{I}_k | -\mathbf{A}^{tr}]$ (untuk kasus biner -**A** = **A**)dengan:

a. **A** merupakan matriks dengan ukuran tetap k x $(n - k)$.

b. $\mathbf{I}_k$ merupakan matriks identitas dengan ukuran k x k (Mac Williams & Sloane 1983).

Berdasarkan pengertian tersebut maka kode didefinisikan sebagai:

$C = \{ x \in (F_q)^n \mid x = \boldsymbol{u}\mathbf{G}, \boldsymbol{u} \in (F_q)^k \}$.

Contoh:

Misalkan $\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$ dan pesan $\boldsymbol{u}$ = (11) maka:

$x = \boldsymbol{u}\mathbf{G}$

$x = [1\,1] \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$

$x = [1\,0\,1\,1] + [0\,1\,0\,1]$

$x = [1\,1\,1\,0]$.

Dengan demikian pesan $\boldsymbol{u}$ = (11) memiliki katakode x = (1110).

Matriks **G** dan **H** memiliki hubungan sebagai berikut:

$\mathbf{G}\mathbf{H}^{tr} = 0$ atau $\mathbf{H}\mathbf{G}^{tr} = 0$

2. Kode Siklik

Kode siklik merupakan subkelas dari kode linear. Menurut Shu dan Costello (1983) ada dua alasan kode siklik menjadi kode yang menarik:

1)Proses *encoding* dan komputasi sindrom dapat diimplementasikan dengan mudah menggunakan register geser umpan balik.

2)Memiliki struktur aljabar sehingga memungkinkan pencarian metode dekoding yang praktis.

Menurut Haykin (2001) kode siklik memiliki 2 sifat dasar:

1. *Linearity property* yaitu hasil penjumlahan antara dua katakode adalah katakode.
2. *Cyclic property* yaitu apabila terjadi pergeseran siklik katakode maka akan menghasilkan katakode.

Apabila katakode $c = (c_0, c_1, \ldots, c_{n-1})$ digeser siklikal sebanyak i tempat ke kanan maka hasilnya adalah

$c^{(i)} = (c_{n-i}, c_{n-i+1}, \ldots, c_{n-1}, c_0, c_1, \ldots, c_{n-i-1})$.

Pergeseran siklik sebanyak i tempat ke kanan sama dengan pergeseran siklik sebanyak $n - i$ tempat ke kiri.

Setiap katakode berkorespondensi 1-1 dengan polinomial kode (*code polynomial*) berderajat $\leq n - 1$. Katakode c memiliki kode polinomial

$c(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_{n-1} x^{n-1}$.

Jika $c_{n-1} \neq 0$ maka derajat $c(x) = n - 1$. Jika $c_{n-1} = 0$ maka derajat $c(x) < n - 1$.

Misalkan F adalah sebuah *field*. $F[x]$ didefinisikan sebagai berikut:

$F[x] = \{p(x) = \sum_{i=0}^{k} a_i x^i \mid a_i \in F\}$.

Ring R_n didefinisikan sebagai berikut:

Ring R_n $= F[x] / (x^n - 1)$

$= \{r(x) \in F[x] \mid \text{derajat } r(x) < n\}$.

Berikut ini beberapa konsep aljabar untuk menjelaskan pengertian kode siklik:

- **Definisi 5** (MacWilliams & Sloane 1983) *Sebuah ideal $\wp$ dari R_n merupakan subruang linear dari R_n sehingga:*
  (i) Jika $c(x) \in \wp$ maka $r(x)c(x) \in \wp$ untuk semua $r(x) \in R_n$. Jelasnya, (i) dapat dikatakan:
  (ii) Jika $c(x) \in \wp$ maka $xc(x) \in \wp$.

- **Definisi 6** (MacWilliams & Sloane 1983) *Kode siklik dengan panjang n adalah sebuah ideal dari R_n.*

Principal ideal $\langle g(x) \rangle = \{r(x)g(x) \mid r(x) \in R_n\}$. $g(x)$ disebut polinomial generator (*generator polynomial*) dari *ideal*.

- **Teorema 7** *Misalkan $\wp$ adalah ideal taknol dalam R_n yaitu sebuah kode siklik dengan panjang n.*
  1) Ada sebuah polinomial monic unik g(x) dengan derajat minimum pada $\wp$. Bukti: Lihat MacWilliams dan Sloane (1983) atau Shu dan Costello (1983).
  2) $\wp = \langle g(x) \rangle$, g(x) merupakan polinomial generator dari $\wp$. Bukti: Lihat MacWilliams dan Sloane (1983).
  3) g(x) adalah faktor dari x^n-1. Bukti: Lihat MacWilliams dan Sloane (1983).
  4) Sembarang $c(x) \in \wp$ dapat dituliskan secara unik sebagai $c(x) = f(x)g(x)$ dalam $F[x]$, dimana $f(x) \in F[x]$ berderajat $< n-r$, r = derajat $g(x)$. Dimensi dari $\wp$ adalah $n-r$. Sehingga pesan $f(x)$ dapat diubah menjadi katakode $f(x)g(x)$. Bukti: Lihat MacWilliams dan Sloane (1983).
  5) Jika $g(x) = g_0 + g_1x + g_2x^2 + \ldots + g_rx^r$ maka $\wp$ dibangkitkan (sebagai sebuah subruang F^n) oleh baris-baris matriks generator

$$\mathbf{G} = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & g_r & & 0 \\ & g_0 & g_1 & \cdots & g_{r-1} & g_r & \\ & & & \cdots & \cdots & & \\ 0 & & g_0 & \cdots & \cdots & & g_r \end{bmatrix}$$

$$= \begin{bmatrix} g(x) \\ xg(x) \\ \cdots \\ \cdots \\ x^{n-r-1}g(x) \end{bmatrix}.$$

Bukti: Lihat MacWilliams dan Sloane (1983).

Misalkan $\wp$ adalah kode siklik dengan polinomial generator $g(x)$. Polinomial generator $g(x)$ membagi x^n-1 atas GF(q) dengan

$$g(x) = \prod_{i \in K} (x - \alpha^i),$$

untuk $i \in K$. K merupakan gabungan koset-koset siklotomik. Akar-akar pangkat n dari unsur identitas $\{\alpha^i : i \in K\}$ disebut *zeros* dari kode.

3. Faktor-Faktor $x^n - 1$

Diasumsikan n dan q selalu prima relatif (sehingga untuk kasus biner n adalah ganjil). Ada bilangan *integer* terkecil m sehingga n membagi habis $q^m - 1$. m disebut *multiplicative order q* mod n.

Pemfaktoran $x^n - 1$ terdiri atas:

- *Pemfaktoran x^n - 1 atas GF(q^m).*
  Ada n elemen berbeda $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ dalam GF(q^m) sehingga

$$x^n - 1 = \prod_{i=0}^{n-1} (x - \alpha_i).$$

  GF(q^m) disebut *splitting field* $x^n - 1$.
- *Pemfaktoran x^n - 1 atas GF(q).*
  Secara umum koset siklotomik mod n atas GF(q) berisi s adalah

$$C_s = \{s, sq, sq^2, sq^3, \ldots, sq^{l-1}\},$$

  dengan $sq^l \equiv s \bmod n$, s disebut representatif koset (*coset representatives*) mod n.
  Polinomial minimal dari α^s adalah

$$M^{(s)}(x) = \prod_{i \in C_s} (x - \alpha^i).$$

  $M^{(s)}(x)$ merupakan polinomial *monic* berderajat minimum dengan koefisien-koefisien dari GF(q) mempunyai α^s sebagai akar. Sehingga

$$x^n - 1 = \prod_{s} M^{(s)}(x).$$

4. Kode Hamming Biner

Kode Hamming diciptakan oleh Richard Hamming (1915-1998) di Laboratorium Bell.

Jika α adalah elemen primitif GF(2^m) maka 1, α^1, α^2, ..., α^{2^m-2} berbeda dan dapat direpresentasikan dengan *m-tuple* biner taknol.

Kode Hamming biner H_m dengan parameter [$n = 2^m - 1$, $k = n - m$, $d = 3$] memiliki matriks cek paritas

$$\mathbf{H} = (1, \alpha^1, \alpha^2, \ldots, \alpha^{2^m-2}).$$

Kode Hamming adalah kode *pengoreksi-1-galat* (*single-error-correcting*).

Teorema 8 *Kode Hamming H_m seperti yang didefinisikan sebelumnya merupakan kode siklik dengan generator polinomial $g(x)=M^{(1)}(x)$.* Bukti: Lihat MacWilliams dan Sloane (1983).

5. Kode BCH Pengoreksi-*t*-galat

Teorema 9 *Misalkan $\mathcal{C}$ adalah sebuah kode siklik dengan polinomial generator $g(x)$ sehingga untuk integer $b \geq 0$, $\delta \geq 1$,*

$$g(\alpha^b) = g(\alpha^{b+1}) = \ldots = g(\alpha^{b+\delta-2}) = 0$$

yaitu kode mempunyai string $\delta - 1$ pangkat α secara berurutan sebagai zeros. Maka jarak minimum kode $\geq \delta$. Bukti: Lihat MacWilliams dan Sloane (1983).

Definisi 7 *Kode siklik dengan panjang n atas GF(q) adalah kode BCH dengan designed distance δ jika, untuk integer $b \geq 0$,*

$g(x) = LCM\{M^{(b)}(x), M^{(b+1)}(x), \ldots, M^{(b+\delta-2)}(x)\}$.

Polinomial $g(x)$ adalah polinomial monic dengan derajat minimum atas GF(q) yang memiliki α^b, α^{b+1}, ..., $\alpha^{b+\delta-2}$ sebagai zeros. Karena itu, c ada di dalam kode siklik jika dan hanya jika $c(\alpha^b) = c(\alpha^{b+1}) = \ldots = c(\alpha^{b+\delta-2}) = 0$.

Apabila $b = 1$ maka disebut *narrow sense* kode BCH. Apabila b = tetap maka disebut *nested* kode BCH.

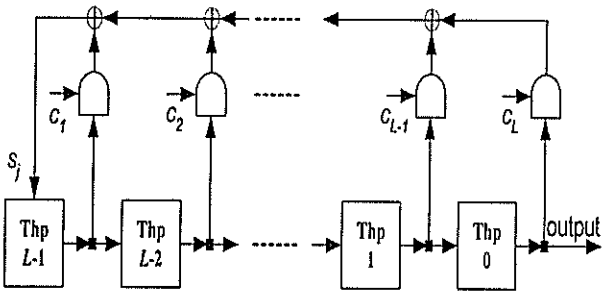
Landasan Teori Untuk *Encoding* Kode BCH

Register Geser Umpan Balik Linear

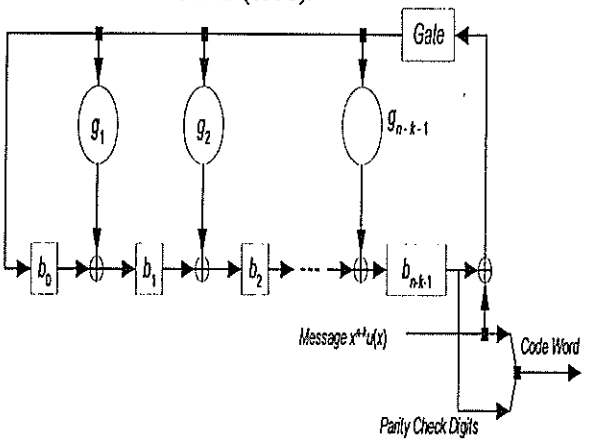
Definisi 8 (Menezes *et al.*, 1996) ***Register geser umpan balik linear*** *(LFSR) dengan panjang L terdiri dari L tahap (stage) yang dinomori dengan 0, 1, 2, ..., L-1, masing-masing mampu menyimpan 1 bit dan mempunyai 1 input dan 1 output; dan dilengkapi suatu clock yang mengontrol pergerakan data. Selama setiap satu satuan waktu operasi berikut dikerjakan:*

1. *isi tahap 0 adalah output dan membentuk bagian dari barisan output;*
2. *isi tahap i digerakan ke tahap i − 1 untuk setiap i, $1 \leq i \leq L-1$; dan*
3. *isi baru dari tahap L − 1 adalah bit umpan balik (feedback bit) s_j yang dihitung dengan menambahkan sekaligus isi subhimpunan tetap dari {0, 1, 2, 3, ..., L - 1} mod 2.*

Skema LFSR diberikan pada Gambar 2. Setengah lingkaran tertutup menunjukan operasi logika AND.



Gambar 2. Skematik LFSR dengan panjang *L* menurut Menezes *et al* (1996).



Gambar 3. Sirkuit *encoding* untuk sebuah kode siklik $\mathcal{C}$ dengan polinomial generator $g(x) = 1 + g_1x + g_2x^2 + \ldots + g_{n-k-1}x^{n-k-1} + x^{n-k}$ menurut Shu dan Costello (1983).

Definisi 9 (Menezes *et al.*, 1996) *LFSR pada Gambar 2 dinotasikan (L, C(D)), dengan*

$C(D) = 1 + c_1D + c_2D^2 + \ldots + c_LD^L \in Z_2[D]$

adalah polinomial penghubung (connection polynomial). LFSR dikatakan non-singular jika derajat C(D) sama dengan L (yaitu $c_L = 1$). Jika isi awal dari tahap i adalah $s_i \in \{0, 1\}$ untuk setiap i, $1 \leq i \leq L-1$, maka $[s_{L-1}, \ldots, s_1, s_0]$ disebut status inisial (initial state) dari LFSR..

Menurut Shu dan Costello (1983), misalkan $g(x)$ adalah polionomial generator dari sebuah kode siklik $\mathcal{C}$. $g(x)$ mempunyai sirkuit *encoding* Gambar 3 yang melakukan langkah-langkah berikut:

1. *Langkah 1.*

Dengan gerbang (*gate*) dinyalakan, *k* digit informasi $u_0, u_1, u_2, \ldots, u_{k-1}$ atau $u(x) = u_0 + u_1x + u_2x^2 + \ldots + u_{k-1}x^{k-1}$ secara simultan digeser ke dalam sirkuit dan saluran komunikasi. Pergeseran pesan $u(x)$ ke dalam sirkuit dimulai dari posisi digit terakhir sama dengan $x^{n-k}u(x)$. Setelah semua pesan masuk ke dalam sirkuit, $n - k$ digit

pada register membentuk sisa (*remainder*) yang disebut digit cek paritas (*parity check digits*).

2. *Langkah 2.*

Putuskan koneksi umpan balik dengan mematikan gerbang.

3. *Langkah 3.*

Geser keluar digit cek paritas dan kirimkan ke saluran. Sebanyak $n - k$ digit cek paritas b_0, b_1, b_2, ..., b_{n-k-1} bersama k digit informasi membentuk sebuah katakode.

METODE PENELITIAN

Desain Proses Konstruksi GF(2^m)

Desain proses konstruksi GF(2^m) meliputi tahapan-tahapan berikut (Gambar 4):

1. **Tentukan nilai *m*.**

   Nilai m akan menentukan polinomial primitif p(x).

2. **Konstruksi elemen-elemen GF(2^m).**

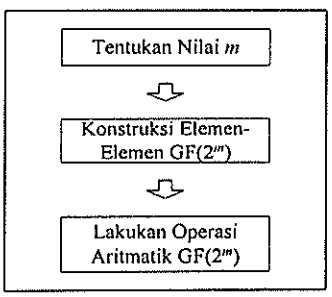
   Apabila GF(2^m) memiliki elemen primitif α dan polinomial primitif p(x) atas GF(2) maka elemen-elemen GF(2^m) akan direpresentasikan dalam bentuk (Tabel 1):

   a. pangkat α,

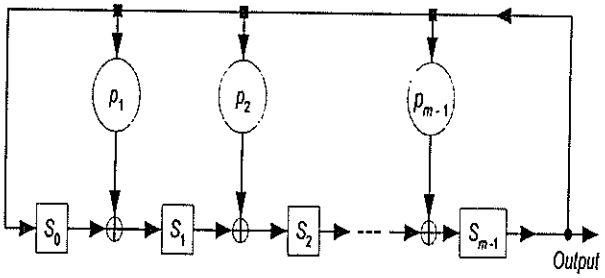
   b. *m-tuple* biner,

   c. simbol i, dengan $0 \le i \le 2^m - 1$.

   Konstruksi elemen-elemen GF(2^m) dilakukan berdasarkan proses kerja register geser umpan balik Gambar 5 dengan status inisial $s = (s_0, s_1, ..., s_{m-1}) = (1, 0, ..., 0, 0)$. Pergeseran dilakukan sebanyak $2^m - 2$ atau berhenti setelah isi register geser arus balik sama dengan status inisial s. Elemen 0, status inisial dan hasil setiap pergeseran membentuk elemen-elemen *m-tuple* biner GF(2^m) (sebelum isi register geser arus balik sama dengan status inisial s). Apabila telah terbentuk elemen-elemen *m-tuple* biner maka dapat disusun elemen-elemen pangkat α dan simbol i, $0 \le i \le 2^m$-1.



Gambar 4. Proses sistematik konstruksi GF(2^m).



Gambar 5. Register geser umpan balik dari polinomial primitif $p(x) = 1 + p_1x + p_2x^2 + ... + p_{m-1}x^{m-1} + x^m$.

Tabel 1. Representasi elemen-elemen GF(2^4) dalam bentuk pangkat α, 4-*tuple* biner dan simbol i, $0 \le i \le 2^m - 1$, polinomial primitif $p(x) = 1 + x + x^4$

Pangkat α	4-*tuple* biner	Simbol i
0	0000	0
1	1000	1
α	0100	2
α^2	0010	3
α^3	0001	4
α^4	1100	5
α^5	0110	6
α^6	0011	7
α^7	1101	8
α^8	1010	9
α^9	0101	10
α^{10}	1110	11
α^{11}	0111	12
α^{12}	1111	13
α^{13}	1011	14
α^{14}	1001	15

3. **Lakukan operasi aritmatik GF(2^m).**

   Operasi aritmatik dalam GF(2^m) dibatasi pada:

   a. *Operasi penjumlahan.*

   Operasi penjumlahan GF(2^m) dilakukan dengan cara melakukan operasi XOR untuk setiap posisi digit *m-tuple* biner.

   b. *Operasi perkalian.*

   Operasi perkalian GF(2^m) dilakukan dengan cara menjumlahkan pangkat α *mod* 2^m-1.

   c. *Operasi invers.*

Operasi invers GF(2^m) dilakukan dengan cara mengurangkan $2^m - 1$ dengan representasi elemen pangkat α. Invers dari α^i adalah α^{n-i}, dengan $n = 2^m - 1$, $1 \le i \le n$.

d. *Operasi pembagian.*

Misalkan a dan b merupakan representasi elemen simbol i. Operasi pembagian GF(2^m) dilakukan dengan cara mengalikan a dengan invers b. Operasi yang dimaksudkan adalah operasi perkalian dan invers pada GF(2^m).

Input bilangan *integer* positif untuk setiap operasi merupakan representasi elemen dalam bentuk simbol i, $0 \le i \le 2^m - 1$. Untuk *input integer* negatif sama dengan *input integer* positif.

Desain Proses Konstruksi Kode BCH

Langkah-langkah konstruksi kode BCH dapat disusun secara sistematik (Gambar 7) sebagai berikut:

1. **Panjang kode BCH *n*.**

   Input n menentukan nilai n pada $x^n + 1$. Seperti yang telah dijelaskan sebelumnya, n dan q (untuk penelitian ini $q = 2$) diasumsikan selalu prima relatif, sehingga n adalah ganjil. Apabila n adalah genap maka akan dicari sedemikian sehingga n menjadi ganjil.

   Misalkan d adalah *integer* ganjil positif, s adalah *integer* genap positif 2^j, $j \ge 0$. Berdasarkan *Lemma* 1, jika panjang kode n adalah genap maka $x^n + 1 = (x^d + 1)^s$. Sehingga *input* panjang kode BCH $n = d$.

   Nilai n menentukan nilai m. Nilai m dicari sedemikian sehingga n membagi habis $2^m - 1$. Misalkan $p(x)$ adalah polinomial primitif. Nilai m merupakan derajat dari $p(x)$.

2. **Konstruksi GF(2^m).**

   Proses konstruksi GF(2^m) dilakukan seperti desain proses sebelumnya.

3. **Konstruksi koset siklotomik.**

   Koset siklotomik atas GF(2) yang memiliki s sebagai representatif koset mod n adalah

   $$C_s = \{s, 2s, 2^2s, 2^3s, ..., 2^{l-1}s\},$$

   dengan $2^l s \equiv s \bmod n$, untuk l adalah *integer* positif.

4. **Konstruksi polinomial minimal.**

   Apabila telah didapatkan C_s pada langkah 4 maka

   $$M^{(s)}(x) = \prod_{i \in C_s} (x + \alpha^i).$$

   Sehingga untuk setiap koset siklotomik akan memiliki polinomial $M^{(s)}(x)$ yang unik. Operasi penjumlahan dan perkalian pembentukan $M^{(s)}(x)$ menggunakan operasi penjumlahan dan perkalian pada GF(2^m).

5. **Konstruksi polinomial generator.**

   Polinomial generator kode BCH adalah

   $g(x) = \text{LCM}\{M^{(b)}(x), M^{(b+1)}(x), ..., M^{(b+\delta-2)}(x)\}$.

   Apabila telah didapatkan polinomial generator maka dapat diketahui parameter kode BCH berikut:

   a. *Designed distance.*

   Designed distance diperoleh berdasarkan Teorema 9 dan koset-koset siklotomik yang diperoleh pada langkah 3. Elemen-elemen koset siklotomik akan digabungkan dan diurutkan menjadi elemen-elemen terurut pangkat α sebagai *zeros*.

   b. *Dimensi kode.*

   Dimensi kode = panjang kode n – derajat $g(x)$. Derajat $g(x)$ dapat diperoleh berdasarkan teorema 4, dengan e adalah banyaknya elemen dari gabungan koset-koset siklotomik.

Desain Proses *Encoding* Kode BCH

Proses *encoding* kode BCH melalui tahapan berikut (Gambar 8):

1. **Tentukan polinomial generator g(*x*).**

   Polinomial generator yang didapatkan sebelumnya akan menentukan bentuk *sirkuit encoding* Gambar 3. Banyaknya sel sirkuit *encoding* merupakan banyaknya simbol cek.

2. **Encoding kode BCH.**

   Proses *encoding* kode BCH dilakukan berdasarkan proses kerja sirkuit *encoding* menurut Shu dan Costello (1983) (Gambar 3). Dalam penelitian ini sebuah katakode direpresentasikan oleh Gambar 6. *Redundant checking part* merupakan simbol cek, sedangkan *message part* merupakan simbol pesan.

   Selama proses *encoding* berlangsung dapat dihitung parameter kode BCH berikut:

   • *Actual distance.*

   Actual distance yang dimaksudkan adalah jarak minimum kode BCH. Untuk memperoleh *actual distance*, diperlukan *input* semua kemungkinan pesan yang bisa di*encoding*. Banyaknya pesan tersebut adalah 2^k. Proses penghitungan *actual distance* dilakukan dengan membandingkan bobot masing-masing katakode. Berdasarkan Teorema 5 maka *actual distance* merupakan bobot yang paling minimum dari katakode.

Implementasi Program

Implementasi setiap tahapan konstruksi kode BCH meliputi hal-hal berikut:

1. *Spesifikasi perangkat keras.*

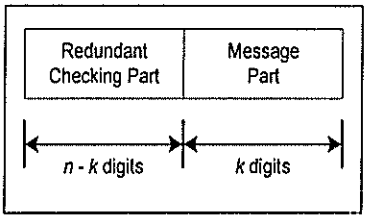
Perangkat keras yang digunakan dalam penelitian adalah:

- Prosesor AMD Athlon(tm) XP 1600+ 1,41 GHz.
- Memori 128 MB.
- *Hard disk* kapasitas 40 GB.

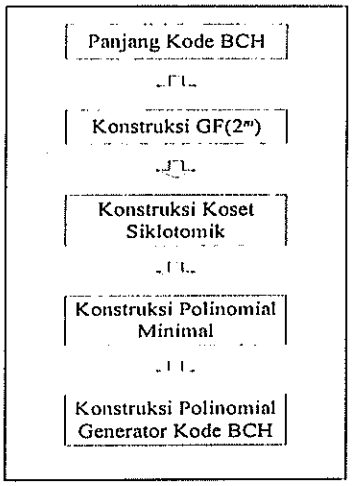
2. *Spesifikasi perangkat lunak.*

Perangkat lunak yang digunakan dalam penelitian adalah:

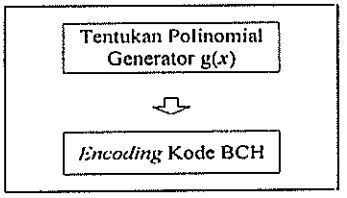
- OS Microsoft Windows XP.
- Matlab versi 6.5.
- Maple versi 8.0.



Gambar 6. Format sistematik sebuah katakode menurut Shu & Costello (1983).



Gambar 7. Proses sistematik konstruksi kode BCH.



Gambar 8. Proses sistematik *encoding* kode BCH.

HASIL DAN PEMBAHASAN

Algoritma dan Implementasi Konstruksi GF(2^m)

a. Konstruksi Elemen-Elemen GF(2^m)

Berikut ini adalah algoritma konstruksi elemen-elemen GF(2^m):

1. Tentukan nilai m.
2. Cari sebuah polinomial primitif $p(x)$ berderajat m. Polinomial primitif $p(x)$ ini menentukan bentuk register geser umpan balik Gambar 5.
3. Dapatkan posisi XOR register geser umpan balik.
4. Inisialisasi register geser umpan balik $\boldsymbol{s} = (s_0, s_1, \ldots, s_{m-1}) = (1, 0, \ldots, 0, 0)$.
5. Lakukan pergeseran 1 bit ke kanan secara siklik. Secara bersamaan lakukan operasi XOR untuk setiap posisi XOR yang telah diperoleh langkah 3. Isi dari register geser hasil langkah ini merupakan elemen *m-tuple* biner.
6. Ulangi langkah 5 sampai dengan isi dari register umpan balik = $(s_0, s_1, \ldots, s_{m-1}) = (1, 0, \ldots, 0, 0)$.
7. Elemen 0, status inisial $\boldsymbol{s}$, dan setiap hasil langkah 5 membentuk elemen *m-tuple* biner GF(2^m).
8. Untuk mendapatkan elemen simbol i dilakukan dengan cara merepresentasikan urutan elemen *m-tuple* biner dengan 0, 1, 2, 3, …, 2^m-1.
9. Untuk mendapatkan elemen pangkat α dilakukan dengan cara mengurangi setiap representasi elemen simbol i dengan 1 kecuali simbol $i = 0$.

Dalam implementasinya, algoritma konstruksi elemen-elemen GF(2^m) dibuat menjadi *fungsi GF2* yang algoritmanya dapat dilihat pada algoritma GF2 (Lampiran 1) dan dijelaskan sebagai berikut:

Input algoritma ini adalah nilai m yang merupakan derajat polinomial primitif. Dengan konfigurasi perangkat keras yang telah didefinisikan, implementasi program hanya dapat memroses untuk nilai $1 \le m \le 20$.

Output algoritma ini adalah matriks ***gf_bin*** dengan ukuran 2^m x m mewakili elemen *m-tuple* biner, matriks ***gf_dec*** dengan ukuran 2^m x 1 mewakili elemen simbol i, dengan $0 \le i \le 2^m - 1$,

nilai m, dan matriks baris p representasi koefisien polinomial primitif atas GF(2) *ascending* power. Elemen pangkat α tidak diciptakan secara langsung.

Contoh:

Input : $m = 8$.

Output:

- Lampiran 2 menunjukan *gf_bin* dan gf_dec dalam bentuk tabel.
- m = 8
- p = 1 0 1 1 1 0 0 0 1
  $\Rightarrow$ representasi $p(x) = 1 + x^2 + x^3 + x^4 + x^8$

b. Operasi Aritmatik GF(2^m)

Algoritma operasi aritmatik pada GF(2^m):

1. *Operasi perkalian.*

Berikut ini adalah algoritma operasi perkalian GF(2^m):

1. Ambil *a* dan *b* representasi elemen simbol *i*. Apabila *a* dan *b* negatif maka akan diubah menjadi positif.
2. Apabila *a* = 0 atau *b* = 0 maka hasil operasi kali adalah 0. Selain itu cari representasi elemen pangkat α dari *a* dan *b*. Proses perkalian dilakukan dengan cara menjumlahkan representasi elemen pangkat α mod $2^m - 1$. Hasilnya ditambah 1 untuk mendapatkan representasi simbol *i* yang sesuai.

Implementasi algoritma operasi perkalian GF(2^m) dibuat menjadi *fungsi GF2_Kali* yang algoritmanya dapat dilihat pada algoritma GF2_Kali (Lampiran 3).

Contoh:

Misalkan *a* dan *b* adalah elemen simbol *i* pada GF(2^8). Apabila $a = 2$, $b = 11$ maka *output*nya adalah 12. Untuk simbol $i = 2$ memiliki elemen pangkat $\alpha = 1$ atau mewakili α^1, simbol $i = 11$ memiliki elemen pangkat $\alpha = 10$ atau mewakili α^{10}, sehingga $\alpha^1 \bullet \alpha^{10} = \alpha^{11}$ yang memiliki simbol $i = 11 + 1 = 12$.

2. *Operasi penjumlahan.*

Berikut ini adalah algoritma operasi penjumlahan GF(2^m):

1. Ambil *a* dan *b* representasi elemen simbol *i*. Apabila *a* dan *b* negatif maka akan diubah menjadi positif.
2. Cari representasi elemen *m-tuple* biner dari *a* dan *b*.
3. Proses penjumlahan dilakukan dengan cara melakukan operasi XOR untuk setiap posisi digit *m-tuple* biner *a* dan *b*.
4. Hasil dari langkah 3 diubah menjadi representasi elemen simbol *i* yang sesuai.

Dalam implementasinya, algoritma operasi penjumlahan GF(2^m) dibuat menjadi *fungsi GF2_Tambah* yang algortitmanya dapat dilihat pada algoritma GF2_Tambah (Lampiran 4).

Contoh:

Misalkan *a* dan *b* adalah elemen simbol *i* pada GF(2^8). Apabila $a = 2$, $b = 11$ maka *output*nya adalah 122. Simbol $i = 2$ memiliki elemen biner 01000000, simbol $i = 11$ memiliki elemen biner 00101110 sehingga 01000000 XOR 00101110 = 01101110 yang memiliki simbol $i = 122$.

3. *Operasi invers.*

Berikut ini adalah algoritma operasi invers GF(2^m):

1. Ambil *a* representasi elemen simbol *i*. Apabila *a* negatif maka diubah menjadi positif.
2. Apabila *a* = 0 maka hasil operasi invers adalah inf (infinity). Selain itu cari representasi pangkat α dari *a*. Proses invers dilakukan dengan cara mengurangi ($2^m - 1$) dengan representasi pangkat α dari *a* mod 2^m-1. Hasilnya diubah ke representasi elemen simbol *i* yang sesuai.

Implementasi algoritma operasi invers GF(2^m) dibuat menjadi *fungsi GF2_Invers* yang algoritmanya dapat dilihat pada algoritma GF2_Invers (Lampiran 5).

Contoh:

Misalkan *a* adalah elemen simbol *i* pada GF(2^8). Apabila $a = 11$ maka *output*nya adalah 246. Simbol $i = 11$ memiliki elemen pangkat $\alpha = 10$, sehingga $\{(2^8 - 1) - 10\} \bmod 2^8 - 1 = (255 - 10) \bmod 255 = 245$, yang memiliki elemen simbol $i = 246$.

4. *Operasi pembagian.*

Berikut ini adalah algoritma operasi pembagian GF(2^m):

1. Ambil *a* dan *b* representasi elemen simbol *i*. Apabila *a* dan *b* negatif maka akan diubah menjadi positif.
2. Apabila *a* ≥ 0 dan *b* = 0 maka hasil operasi bagi adalah inf. Apabila *a* = 0 dan *b* > 0 maka hasil operasi bagi adalah 0. Selain itu cari invers *b*. Operasi bagi dilakukan

dengan cara mengalikan *a* dan hasil operasi invers *b* (dalam hal ini operasi perkalian dan invers pada GF(2^m)).

Dalam implementasinya algoritma operasi pembagian GF(2^m) dibuat menjadi *fungsi GF2_Bagi* yang algoritmanya dapat dilihat pada algoritma GF2_Bagi (Lampiran 6).

Contoh:

Misalkan *a* dan *b* adalah elemen simbol *i* pada GF(2^8). Apabila $a = 2$, $b = 11$, *output*nya adalah 247. Simbol $i = 2$ memiliki representasi elemen pangkat $\alpha = 1$, simbol $i = 11$ memiliki representasi elemen pangkat $\alpha = 10$, sehingga

$2 / 11 = 2 \bullet \text{invers}(11) \Rightarrow$ (simbol *i*)
$= 2 \bullet \{[(2^8 - 1) - 10] + 1\} \Rightarrow$ (simbol *i*)
$= 2 \bullet 246 \Rightarrow$ (simbol *i*)
$= 1 + 245 \Rightarrow$ (pangkat α)
$= 246 \Rightarrow$ (pangkat α)
$= 247. \Rightarrow$ (simbol *i*)

Algoritma dan Implementasi Konstruksi Kode BCH

a. Panjang Kode BCH

Berikut ini adalah algoritma untuk menentukan panjang kode ganjil:

1. Masukan panjang kode *n*. Panjang kode *n* harus integer lebih dari 0.
2. Apabila panjang kode adalah genap maka cari panjang kode ganjil yang sesuai menggunakan dengan persamaan $(x^n + 1) = (x^d + 1)^s$, *d* ganjil dan *s* genap dengan $s = 2^j$, $j \geq 0$.

Implementasi algoritma untuk menentukan panjang kode ganjil dibuat menjadi *fungsi Carids* yang algoritmanya dapat dilihat pada algoritma Carids (Lampiran 7).

Contoh:

a. Untuk $n = 23$, $(x^{23} + 1) = (x^{23} + 1)^1$ didapatkan $d = 23$, dan $s = 2^0 = 1$. Dengan demikian $n = 23$.
b. Untuk $n = 24$, $(X^{24} + 1) = (X^3 + 1)^8$ didapatkan $d = 3$, dan $s = 2^3 = 8$. Dengan demikian $n = 3$.

Berikut ini adalah algoritma untuk menentukan nilai *m*:

1. Tentukan nilai *n*.
2. Cari *m* sedemikian sehingga $n | 2^m - 1$.

Implementasi algoritma untuk menentukan nilai *m* dibuat menjadi *fungsi Carim* yang algoritmanya dapat dilihat pada algoritma Carim (Lampiran 8).

Contoh:

Untuk $n = 23$ didapatkan $m = 11$, karena $23 | 2^{11} - 1$.

b. Konstruksi GF(2^m)

Gunakan konstruksi GF(2^m) sebelumnya.

c. Konstruksi Koset Siklotomik

Berikut ini adalah algoritma konstruksi siklotomik:

1. Tentukan nilai *n* (bilangan integer ganjil). Elemen gabungan semua koset adalah {0, 1, 2, 3, ..., n - 1}.
2. $C_0 = \{0\}$. Kurangi elemen gabungan semua koset dengan C_0.
3. Tentukan $s = 1$. Cari C_1. Kurangi sisa elemen gabungan semua koset dengan C_1.
4. Selama masih ada elemen gabungan semua koset, dapatkan *s* berikutnya yaitu elemen terkecil sisa elemen gabungan koset kemudian cari C_s. Kurangi sisa elemen gabungan semua koset dengan C_s.

Implementasi algoritma konstruksi koset siklotomik dibuat menjadi *fungsi Conjugate* dan *Cyclotomic* yang algoritma dijelaskan sebagai berikut:

1. Algoritma Conjugate (Lampiran 9).

Input algoritma ini adalah sebuah elemen koset siklotomik *s* dan panjang kode ganjil *n*, dengan $0 \leq s < n$ dan $n > 0$. Proses dilakukan berdasarkan pembentukan koset siklotomik *mod n* atas GF(2). *Output* algoritma ini adalah matriks baris ***konj*** yang merepresentasikan representatif koset mod *n* dan *conjugate*nya, dan *e* sebuah bilangan *integer* positif terkecil sehingga $2^e s \equiv s \bmod n$.

Contoh:

a. *Input* : $s = 1$, $n = 15$.
   Output :
   • konj = 1 2 4 8
   • e = 4
b. *Input* : $s = 2$, $n = 15$.
   Output :
   • konj = 2 4 8 1
   • e = 4

2. Algoritma Cyclotomic(Lampiran 10).

Input algoritma ini adalah bilangan *integer* ganjil $n > 0$. Peneliti mendefinisikan suatu

matriks *c* dengan elemen-elemen terurut yang merepresentasikan gabungan dari semua koset kecuali $C_0 = \{0\}$.

Dimulai dengan $s = 1$ cari semua *conjugate*nya. Kemudian lakukan operasi pengurangan himpunan antara *c* dan C_1. Ambil elemen pertama hasil pengurangan himpunan tersebut untuk mendapakan *s* berikutnya, proses ini berlangsung selama masih terdapat elemen matriks *c*.

Output dari algoritma Cyclotomic adalah matriks *cs* yang setiap barisnya merepresentasikan koset siklotomik, dan matriks baris *s* yang merepresentasikan representatif koset minimum mod *n* dari setiap baris *cs*. Untuk $n = 1$ hanya akan tercipta $C_0 = \{0\}$.

Contoh:

a. *Input* : $n = 255$.
Output : Dapat dilihat pada Lampiran 11.
Kecuali baris pertama, nilai 0 hanya sebagai tambahan agar setiap baris memiliki banyak kolom yang sama, sehingga elemen-elemen koset adalah adalah elemen taknol pada setiap baris. Misalkan untuk baris ke-10 matriks *cs* mewakili $C_{17} = \{17, 34, 68, 136\}$.

b. *Input* : $n = 23$
Output :

- cs =

0	0	0	0	0	0	0	0	0	0	0
1	2	4	8	16	9	18	13	3	6	12
5	10	20	17	11	22	21	19	15	7	14

- s =

0	1	5

** Untuk memudahkan proses a, b, c peneliti menyusun *fungsi KodeSiklik* yang algoritmanya dapat dilihat pada algoritma KodeSiklik (Lampiran 12). *Input* algoritma ini adalah panjang kode *n*, dengan *n* adalah *integer* > 0. Proses algoritma ini melibatkan algoritma Carids, Carim, GF2, Conjugate, dan Cyclotomic. Selain itu, algoritma ini juga akan mencari pangkat α pembuat siklik. Sebagai contoh hasil algoritma KodeSiklik untuk $n = 31$ dapat dilihat pada Lampiran 13.

d. Konstruksi Polinomial Minimal

Berikut ini adalah algoritma konstruksi polinomial minimal:

1. Tentukan sebuah koset siklotomik C_s untuk *n* (bilangan *integer* ganjil) tertentu. Setiap elemen C_s merupakan akar dari polinomial minimal $M^{(s)}(x)$.
2. Apabila $n = 2^m - 1$ maka $r = 1$ (r adalah pangkat α pembuat siklik). Apabila $n \neq 2^m - 1$ maka dapatkan r dengan cara :
   $(\alpha^r)^n = (\alpha)^{2^m - 1}$
   $rn = 2^m - 1$
   $r = (2^m - 1) / n$.
3. Kalikan setiap elemen C_s dengan r.
4. Polinomial minimal
   $$M^{(s)}(x) = \prod_{i \in C_s} (x - \alpha^i).$$

Elemen-elemen setiap koset siklotomik membentuk akar-akar suatu polinomial minimal, sehingga setiap koset siklotomik memiliki polinomial minimal yang unik.

Misal:

- Untuk $n = 255$ didapatkan $m = 8$ maka $r = (2^8 - 1)/255 = 1$. Sehingga untuk $C_{17} = \{17, 34, 68, 136\}$ memiliki polinomial minimal
  $M(x) = (x + \alpha^{17})(x + \alpha^{34})(x + \alpha^{68})(x + \alpha^{136})$.

- Untuk $n = 23$ didapatkan $m = 11$ maka $r = (2^{11} - 1) / 23 = 89$. Sehingga untuk $C_1 = \{1, 2, 4, 8, 16, 9, 18, 13, 3, 6, 12\}$ memiliki polinomial minimal
  $M(x) = (x + \alpha^{89})(x + \alpha^{178})(x + \alpha^{356})(x + \alpha^{712})(x + \alpha^{1424})(x + \alpha^{801})(x + \alpha^{1602})(x + \alpha^{1157})(x + \alpha^{267})(x + \alpha^{534})(x + \alpha^{1068})$.

Fungsi Matriks_MinPoly_XR dengan algoritma Matriks_MinPoly_XR (Lampiran 17) dan *fungsi Matriks_MinPoly_X* dengan algoritma Matriks_MinPoly_X (Lampiran 18), merupakan implementasi algoritma konstruksi polinomial minimal.

Input kedua algoritma tersebut adalah matriks baris *s_cs* yang merepresentasikan elemen-elemen koset yang akan dicari polinomial minimalnya dan panjang kode ganjil *n*, dengan $0 \leq s\_cs < n$. Setiap elemen *s_cs* dicari *conjugate*nya. Kemudian polinomial minimal diciptakan menggunakan operasi-operasi pada $GF(2^m)$. Apabila ada beberapa elemen memiliki koset sama maka hanya akan direpresentasikan oleh 1 polinomial minimal.

Yang membedakan kedua algoritma ini adalah penggunaan polinomial *reciprocal* pada algoritma Matriks_MinPoly_XR. Algoritma ini sekaligus menciptakan polinomial *reciprocal* setiap polinomial minimal.

Output kedua algoritma ini adalah matriks yang setiap barisnya merepresentasikan s, $M^{(s)}(x)=\prod_{i\in C_s}(x+\alpha^i)$. Khusus untuk polinomial *reciprocal*, nilai s mewakili representatif koset minimum mod n.

Contoh:

1. *Input*: $s\_cs = 17$, $n = 255$.
   - *Output* Matriks_MinPoly_XR:
     17, 1 + X + X^4
     119, 1 + X^3 + X^4 (polinomial *reciprocal*)
   - *Output* Matriks_MinPoly_X =
     17, 1 + X + X^4
2. *Input* : s_cs = [1, 3, 5, 7, 9, 15, 30, 60, 16], $n = 255$.
   - *Output* Matriks_MinPoly_XR =
     1, 1 + X^2 + X^3 + X^4 + X^8
     127, 1 + X^4 + X^5 + X^6 + X^8
     3, 1 + X + X^2 + X^4 + X^5 + X^6 + X^8
     63, 1 + X^2 + X^3 + X^4 + X^6 + X^7 + X^8
     5, 1 + X + X^4 + X^5 + X^6 + X^7 + X^8
     95, 1 + X + X^2 + X^3 + X^4 + X^7 + X^8
     7, 1 + X^3 + X^5 + X^6 + X^8
     31, 1 + X^2 + X^3 + X^5 + X^8
     9, 1 + X^2 + X^3 + X^4 + X^5 + X^7 + X^8
     111, 1 + X + X^3 + X^4 + X^5 + X^6 + X^8
     15, 1 + X + X^2 + X^4 + X^6 + X^7 + X^8

   Untuk 1 dan 16 sama-sama memiliki koset siklotomik C_1 sehingga hanya diwakili $M^{(1)}(x)$. Begitu pula untuk 15, 30, dan 60 sama-sama memiliki koset siklotomik C_{15} sehingga hanya diwakili $M^{(15)}(x)$.

   - *Output* Matriks_MinPoly_X:
     1, 1 + X^2 + X^3 + X^4 + X^8
     3, 1 + X + X^2 + X^4 + X^5 + X^6 + X^8
     5, 1 + X + X^4 + X^5 + X^6 + X^7 + X^8
     7, 1 + X^3 + X^5 + X^6 + X^8
     9, 1 + X^2 + X^3 + X^4 + X^5 + X^7 + X^8
     15, 1 + X + X^2 + X^4 + X^6 + X^7 + X^8
3. *Input*: $s\_cs = 8$, $n = 23$.
   - *Output* Matriks_MinPoly_XR:
     8, 1 + X + X^5 + X^6 + X^7 + X^9 + X^11
     5, 1 + X^2 + X^4 + X^5 + X^6 + X^10 + X^11 (reciprocal polinomial)
   - *Output* Matriks_MinPoly_X:
     8, 1 + X + X^5 + X^6 + X^7 + X^9 + X^11

Kasus khusus. Kasus khusus terjadi ketika hanya terbentuk koset siklotomik C_0 dan C_1. Untuk kasus tersebut $x^n + 1 = (1 + x)(1 + x + x^2 + \ldots + x^{n-1})$.

Contoh:

1. *Input* : s_cs = [0, 1], $n = 11$.
   - *Output* kedua algoritma:
     0, 1 + X
     1, 1 + X + X^2 + X^3 + X^4 + X^5 + X^6 + X^7 + X^8 + X^9 + X^10
2. *Input* : s_cs = [2, 3, 4], $n = 11$.
   - *Output* kedua algoritma:
     1, 1 + X + X^2 + X^3 + X^4 + X^5 + X^6 + X^7 + X^8 + X^9 + X^10

Untuk 2, 3, dan 4 memiliki koset siklotomik C_1 sehingga hanya diwakili $M^{(1)}(x)$.

e. Konstruksi Polinomial Generator

Berikut ini adalah algoritma konstruksi polinomial generator kode BCH:

1. Urutkan pangkat α sebagai *zeros* sehingga membentuk susunan α^b, α^{b+1}, ..., $\alpha^{b+\delta-2}$.
2. Dapatkan urutan pangkat α sebagai *zeros* terbanyak untuk memperoleh designed distance δ.
3. Polinomial generator $g(x)=\mathrm{LCM}\{M^{(b)}(x), M^{(b+1)}(x), \ldots, M^{(b+\delta-2)}(x)\}$.
4. Dapatkan derajat polinomial generator $g(x)$ untuk menghitung dimensi kode BCH.

Untuk menciptakan polinomial generator peneliti menggunakan Maple 8.0, hal ini berkaitan dengan penggunaan *fungsi LCM* untuk faktorisasi polinomial.

Implementasi algoritma konstruksi polinomial generator kode BCH meliputi *fungsi Eksp*, *Minpoly_Generator*, *Polinomial_Generator*, dan *BCH_List* yang algoritmanya dijelaskan sebagai berikut:

a. Algoritma Eksp (Lampiran 19).

Algoritma ini membutuhkan *input* berupa matriks baris ***gbngn*** yang merepresentasikan elemen-elemen pangkat α. Elemen-elemen tersebut akan diurutkan sehingga membentuk urutan pangkat α sebagi *zeros* dengan bentuk b, $b+1$, $b+2$, ..., $b+\delta-2$. *Output* dari algoritma ini adalah matriks ***s_g*** yang merepresentasikan urutan pangkat α, ***bd*** yang merepresentasikan nilai b, ***dd*** yang mrepresentasikan *designed distance* maksimum, serta semua kemungkinan *designed distance* yang tercipta ***ddl***.

Contoh:

1. *Input* : ***gbngn*** = [1, 2, 3, 4, 5, 7, 8, 9, 11, 14, 15, 16].
   Output:
   o dd = 6
   o s_g = 1 2 3 4 5
   o bd = 1
   o ddl = 6 4 2 4

2. *Input* : ***gbngn*** = [1, 3, 4, 5, 7, 8, 9, 11, 14, 15, 16].
   Output:
   o dd = 4
   o s_g =
   3 4 5
   7 8 9
   14 15 16
   o bd = 3 7 14
   o ddl = 2 4 4 2 4

   Setiap baris matriks ***bd*** merepresentasikan urutan pangkat α yang memiliki *designed distance* sama.

b. Algoritma Minpoly_Generator (Lampiran 20).

Algoritma ini akan mencari polinomial minimal yang dimiliki masing-masing elemen urutan pangkat α. Apabila ada beberapa elemen pangkat α memiliki koset siklotomik sama maka hanya akan diwakili oleh satu polinomial minimal.

Input algoritma adalah panjang kode n dan matriks ***s_cs*** yang setiap barisnya merepresentasikan urutan b, $b+1$, $b+\delta-2$ dari polinomial generator
$g(x) = \text{LCM}\{M^{(b)}(x), M^{(b+1)}(x), ..., M^{(b+\delta-2)}(x)\}$.

Output program adalah matriks ***s_mp_zeros*** yang setiap barisnya merepresentasikan representatif koset mod n dari polinomial-polinomial minimal pembentuk polinomial generator kode BCH dan matriks kolom ***der_g*** yang merepresentasikan derajat polinomial generator. Dengan memanfaatkan matriks ***s_mp_zeros*** maka dapat ditentukan polinomial-polinomial minimal yang akan mengonstruksi polinomial generator kode BCH.

Contoh:

- *Input* : ***s_cs*** = [1, 2, 3, 4, 5], ***n*** = 255.
  Output :
  o s_mp_zeros = 1 3 5
  o der_g = 24

Proses diatas menunjukan bahwa untuk $g(x) = \text{LCM}\{M^{(1)}(x), M^{(2)}(x), M^{(3)}(x), M^{(4)}(x), M^{(5)}(x)\} = \text{LCM}\{M^{(1)}(x), M^{(3)}(x), M^{(5)}(x)\}$ karena $M^{(1)}(x) = M^{(2)}(x) = M^{(4)}(x)$.

- *Input* : ***s_cs*** = [3, 4, 5; 7, 8, 9; 14, 15, 16], ***n*** = 255.
  Output :
  o s_mp_zeros =
  3 1 5
  7 1 9
  7 15 1
  o der_g =
  24
  24
  24

c. Algoritma Polinomial_Generator (Lampiran 21).

Merupakan algoritma untuk mencari polinomial generator kode BCH. Program diimplementasikan menggunakan Maple 8.0.

Input dari algoritma ini adalah polinomial-polinomial minimal dalam x dengan koefisien atas GF(2) dari urutan pangkat α sebagai *zeros* (berdasarkan ***s_mp_zeros*** hasil algoritma Minpoly_Generator). *Output* dari algoritma ini adalah matriks baris yang merepresentasikan koefisien-koefisien polinomial generator *ascending power* atas GF(2).

Contoh:

Untuk n = 255, kode memiliki representatif koset minimum mod 255 dan polinomial-polinomial minimal sebagai berikut:

0, 1 + X
1, 1 + X^2 + X^3 + X^4 + X^8
3, 1 + X + X^2 + X^4 + X^5 + X^6 + X^8
5, 1 + X + X^4 + X^5 + X^6 + X^7 + X^8
7, 1 + X^3 + X^5 + X^6 + X^8
9, 1 + X^2 + X^3 + X^4 + X^5 + X^7 + X^8
11, 1 + X + X^2 + X^5 + X^6 + X^7 + X^8
13, 1 + X + X^3 + X^5 + X^8
15, 1 + X + X^2 + X^4 + X^6 + X^7 + X^8
17, 1 + X + X^4
19, 1 + X^2 + X^5 + X^6 + X^8
21, 1 + X + X^3 + X^7 + X^8
23, 1 + X + X^5 + X^6 + X^8
25, 1 + X + X^3 + X^4 + X^8
27, 1 + X + X^2 + X^3 + X^4 + X^5 + X^8
29, 1 + X^2 + X^3 + X^7 + X^8
31, 1 + X^2 + X^3 + X^5 + X^8
37, 1 + X + X^2 + X^3 + X^4 + X^6 + X^8
39, 1 + X^3 + X^4 + X^5 + X^6 + X^7 + X^8
43, 1 + X + X^6 + X^7 + X^8
45, 1 + X^3 + X^4 + X^5 + X^8
47, 1 + X^3 + X^5 + X^7 + X^8
51, 1 + X + X^2 + X^3 + X^4
53, 1 + X + X^2 + X^7 + X^8

55, 1 + X^4 + X^5 + X^7 + X^8
59, 1 + X^2 + X^3 + X^6 + X^8
61, 1 + X + X^2 + X^3 + X^6 + X^7 + X^8
63, 1 + X^2 + X^3 + X^4 + X^6 + X^7 + X^8
85, 1 + X + X^2
87, 1 + X + X^5 + X^7 + X^8
91, 1 + X^2 + X^4 + X^5 + X^6 + X^7 + X^8
95, 1 + X + X^2 + X^3 + X^4 + X^7 + X^8
111, 1 + X + X^3 + X^4 + X^5 + X^6 + X^8
119, 1 + X^3 + X^4
127, 1 + X^4 + X^5 + X^6 + X^8.

Misalkan untuk *input* = $M^{(1)}(x)$, $M^{(3)}(x)$, $M^{(5)}(x)$, $M^{(7)}(x)$ maka *output* dari algoritma Polinomial_Generator adalah [1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1] yang merepresentasikan polinomial generator

$g(x) = 1 + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^9 + x^{14} + x^{16} + x^{17} + x^{19} + x^{20} + x^{22} + x^{25} + x^{27} + x^{26} + x^{29} + x^{31} + x^{30} + x^{32}$.

d. Algoritma BCH_List (Lampiran 22).

Merupakan algoritma untuk menentukan *designed distance*, polinomial generator, dan dimensi kode BCH *narrow sense*. Algoritma ini melibatkan aloritma Carids, Cyclotomic, Eksp, dan Minpoly_Generator.

Sebagai *input* algoritma ini adalah panjang kode BCH ***n***, dengan ***n*** adalah *integer* positif > 0. Apabila ***n*** adalah bilangan genap maka dicari nilai ***n*** ganjil.

Output dari algoritma ini adalah matriks representasi *designed distance* ***dd***, matriks representasi indeks polinomial-polinomial minimal pembentuk polinomial generator ***s_mp***, matriks representasi dimensi kode masing-masing polinomial generator ***dim***, dan matriks ***list*** representasi gabungan antara ***dd***, ***s_mp***, dan ***dim***. Setiap baris pertama matriks ***s_mp*** merepresentasikan $g(x) = 1$.

Contoh:

1. *Input* : ***n*** = 15 (ganjil).
   Output:
   - dd =

     1 0 0 0
     3 0 0 0
     5 0 0 0
     7 0 0 0
     9 11 13 15
   - s_mp =

     1 0 0 0
     1 0 0 0
     1 3 0 0
     1 3 5 0
   - dim =

     1 3 5 7
   - dim =

     15
     11
     7
     5
     1
   - list =

     1 0 0 0 1 0 0 0 15
     3 0 0 0 1 0 0 0 11
     5 0 0 0 1 3 0 0 7
     7 0 0 0 1 3 5 0 5
     9 11 13 15 1 3 5 7 1

Nilai 0 pada matriks ***dd*** dan ***s_mp*** adalah nilai tambahan agar setiap baris memiliki banyak kolom yang sama, bukan merepresentasikan *designed distance* 0. Hasil matriks ***list*** merepresentasikan Tabel 2.

Tabel 2. *Designed distance*, polinomial generator, dimensi kode BCH *narrow sense* dengan panjang $n = 15$

Designed Distance	**Polinomial Generator**	**Dimensi**
1	1	15
3	$M^{(1)}(x)$	11
5	$M^{(1)}(x)M^{(3)}(x)$	7
7	$M^{(1)}(x)M^{(3)}(x)M^{(5)}(x)$	5
9, 11, 13, atau 15	$M^{(1)}(x)M^{(3)}(x)M^{(5)}(x)M^{(7)}(x)$	1

2. *Input* : ***n*** = 16 (genap).
   Output :
   - dd = 1
   - s_mp = 1
   - dim = 1
   - list = 1 1 1

Untuk $n = 16$ memiliki panjang kode ganjil $n = 1$, $(x^{16} + 1) = (x^1 + 1)^{16}$. Koset siklotomik yang terbentuk hanya $C_0 = \{0\}$. Hasil matriks ***list*** merepresentasikan Tabel 3.

Tabel 3. *Designed distance*, polinomial generator, dimensi kode BCH *narrow sense* dengan panjang $n = 16$

Designed Distance	**Polinomial Generator**	**Dimensi**
1	1	1

Algoritma dan Implementasi *Encoding* Kode BCH

Berikut ini adalah algoritma proses *encoding* kode BCH:

1. Ambil suatu polinomial generator $g(x)$ dan siapkan pesan yang akan di*encoding*. Polinomial generator $g(x)$ ini menentukan bentuk sirkuit *encoding* Gambar 3.
2. Dapatkan posisi XOR sirkuit *encoding*.
3. Inisialisasi sirkuit *encoding* $b = (b_0, b_1, b_2, \ldots, b_{n-k-1}) = (0, 0, 0, \ldots, 0)$.
4. Untuk setiap pesan lakukan:
   a. Masukan setiap satu digit pesan ke dalam sirkuit *encoding* dimulai dari posisi digit terakhir.
   b. Lakukan operasi XOR digit pesan dengan isi sel b_{n-k-1}.
   c. Lakukan pergeseran 1 bit ke kanan secara siklik. Secara bersamaan lakukan operasi XOR untuk setiap posisi XOR yang diperoleh langkah 2.
   d. Ulang langkah a, b, c sampai dengan semua digit pesan masuk ke dalam sirkuit *encoding*. Isi dari sirkuit *encoding* ditambahkan di bagian depan pesan membentuk katakode.
5. Hitung bobot katakode. Apabila pesan adalah semua kemungkinan pesan yang dapat di*encoding* kode BCH maka actual distance didapatkan dengan cara membandingkan setiap bobot katakode untuk memperoleh bobot minimum.

Implementasi algoritma proses *encoding* kode BCH meliputi *fungsi LFSR*, *Bobot*, dan *Encoding_BCH* yang algoritmanya dijelaskan sebagai berikut:

a. Algoritma LFSR (Lampiran 23).

Algoritma ini digunakan untuk menentukan posisi XOR sirkuit *encoding*. *Input* algoritma ini adalah matriks baris ***g*** yang merepresentasikan koefisien-koefisien polinomial generator *ascending power* atas GF(2). *Output* algoritma ini adalah nilai derajat polinomial generator ***der_g*** dan matriks baris ***pxor*** yang merepresentasikan posisi XOR yang dimiliki oleh sirkuit *encoding*.

Contoh:

- *Input* : ***g*** = [1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1] (Untuk $n = 255$, dengan polinomial-polinomial minimal yang terdefinisi sebelumnya).

Output:

- o pxor = 2 3 4 5 6 7 9 14 16 17 19 20 22 25 26 27 29 30 31 32.
- o der_g = 32

Bentuk sirkuit *encoding* dari *input* di atas dapat dilihat pada Lampiran 24. Polinomial generator *input* diatas menambahkan 32 bit simbol cek pada simbol pesan dengan dimensi 255 – 32 = 223 bit.

b. Algoritma Bobot (Lampiran 25).

Merupakan algoritma untuk mencari bobot suatu katakode. *Input* algoritma ini adalah matriks baris ***katakode*** yang merepresentasikan sebuah katakode atas GF(2^m). Matriks ***bbt*** merepresentasikan banyaknya elemen taknol dari ***katakode*** merupakan *output* algoritma ini.

Contoh:

- *Input* : ***katakode*** = [0 1 0 1 0 0 0 0 1 1 1 0 1 1 0]

Output :

- o bbt = 7

c. Algoritma Encoding_BCH (Lampiran 26).

Merupakan algoritma untuk melakukan proses *encoding* kode BCH. Algoritma ini membutuhkan pesan yang akan di*encoding* ***psn***, panjang kode BCH ***n***, dan polinomial generator ***g*** sebagai *input*. Pesan yang akan di*encoding* merupakan *bitstring* dengan panjang sama dengan dimensi kode. *Output* algoritma ini adalah ***katakode***, bobot minimum ***jm***, dan posisi XOR LFSR ***pxor***. Apabila pesan yang di*encoding* hanya ada satu maka bobot minimum merupakan bobot katakode, sedangkan apabila pesan yang di*encoding* lebih dari satu maka bobot minimum merupakan bobot minimum dari katakode-katakode yang dihasilkan. *Actual distance* merupakan bobot minimum semua kemungkinan katakode.

Contoh:

- *Input* : ***psn*** = [1, 0, 1, 1, 0], ***n*** = 15, ***g*** = [1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1].

Output:

- o katakode = 0 1 0 1 0 0 0 0 1 1 1 0 1 1 0
- o jm = 7 (bobot)
- o pxor = 1 2 4 5 8 10

- *Input* : ***psn*** = [1, 0, 1, 1, 0; 0, 0, 0, 0, 1], ***n*** = 15, ***g*** = [1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1].

Output:

- o katakode = 0 1 0 1 0 0 0 0 1 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 1 0 0 0 0 1

- jm = 7 (bobot minimum)
- pxor = 1 2 4 5 8 10

Secara lengkap Lampiran 27 menampilkan semua kemungkinan katakode untuk n = 15, dimensi = 5 dengan polinomial generator $g(x) = 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}$.

KESIMPULAN DAN SARAN

Kesimpulan

Dari pembahasan dapat ditarik kesimpulan sebagai berikut :

1. Panjang kode BCH n selalu dalam bentuk ganjil. Apabila diinginkan kode BCH dengan panjang genap maka panjang kode tersebut harus didapatkan panjang kode ganjil yang sesuai.
2. GF(2^m) memiliki peran yang sangat penting dalam konstruksi kode BCH. GF(2^m) harus diciptakan terlebih dulu karena operasi-operasinya dibutuhkan untuk proses pembentukan polinomial minimal. Untuk menciptakan GF(2^m) diperlukan sebuah polinomial primitif dengan koefisien-koefisien atas GF(2).
3. Setiap koset siklotomik memiliki polinomial minimal yang unik. Dengan memanfaatkan proses perkalian dan penjumlahan GF(2^m), pangkat α pembuat siklik, dan setiap elemen koset maka dapat dibentuk polinomial minimal.
4. Setiap polinomial generator $g(x)$memiliki *designed distance* tertentu, menentukan besarnya dimensi kode, menentukan besarnya *actual distance*, menentukan bentuk sirkuit *encoding*, dan menentukan banyaknya bit simbol cek yang ditambahkan pada simbol pesan.

Saran

Semua algoritma yang disusun oleh penulis merupakan algoritma-algoritma yang memudahkan untuk konstruksi elemen dan operasi GF(2^m), konstruksi kode BCH, dan proses *encoding* kode BCH. Dalam hal ini penulis tidak mengatakan bahwa implementasi dari algoritma-algoritma tersebut merupakan implementasi terbaik. Dimungkinkan algoritma-algoritma tersebut diimplementasikan menggunakan konsep pemrograman berorientasi objek.

Dalam penelitian ini penulis belum menyertakan proses *decoding* kode BCH. Proses ini dapat digunakan untuk penelitian selanjutnya.

DAFTAR PUSTAKA

Guritman, S. 2003. Pemodelan Kode Linear. Pelatihan Pemodelan Matematika: Pengembangan dan Implementasinya dalam Komputer, Bogor 4 – 16 Agustus 2003.

Haykin, S. 2001. Communication Systems 4th Edition. John Wiley & Sons, Inc.

Hill, R. 1986. A First Course in Coding Theory. Oxford Applied Mathematics and Computing Science Series.

MacWilliams FJ, Sloane NJA. 1983. The Theory of Error Correcting Codes. North-Holland Mathematical Library.

Menezes A, Van Oorschot P, Vanstone P. 1996. Handbook Of Applied Cryptography. CRC Press.

Pless, V. 1989. Introduction to The Theory of Error Correcting Codes Second Edition. John Wiley and Sons, A Wiley-Interscience Publication.

Shu L, Costello DJ JR. 1983. Error Control Coding Fundamentals and Applications. Prentice-Hall, Inc. Englewood Cliffs, New Jersey 07632.

LAMPIRAN

Lampiran 1. Algoritma GF2

Algoritma GF2

'Algoritma untuk menciptakan elemen-elemen GF(2^m)'

```
DEKLARASI
'variabel global akan memudahkan fungsi operasi-operasi GF(2^m)'
mtuple       : integer 'derajat polinomial primitif p(x)'
global m     : integer 'derajat polinomial primitif p(x)'
global gf_dec: array [1..2^m,1]of integer 'representasi elemen simbol i, 0 ≤
               i ≤ 2^m - 1'
global gf_bin: array [1..2^m,1..m]of integer 'representasi elemen m-tuple
               biner'
p            : array [1..m+1] of integer 'koefisien p(x) atas GF(2) ascending
               power'
init         : array [1..m] of integer 'representasi register geser umpan
               balik'
gf           : array [2^m,1..m+1] of integer 'representasi elemen biner dan
               simbol i'
pos_trkhr    : integer 'elemen terakhir register geser XOR input pesan'

DESKRIPSI
read(mtuple);
if mtuple bukan integer ≥ 1 then
   program berhenti dengan pesan kesalahan;
else
   m ← mtuple;
   init ← [1, 0, 0, …,0];'inisialisasi isi register geser umpan balik'
   gf ← array dengan elemen 0; 'gf[1,1..m+1] merupakan elemen 0 pada GF(2^m)'
   gf[2,1..m] ← init;
   gf[2,m+1] ← 1;
   p ← cari polinomial primitif berderajat m;
   dapatkan posisi XOR register geser umpan balik p(x);
   for i ← 3 to 2^m do
      pos_trkhr ← init[m]; 'dapatkan elemen terakhir init'
      geser secara siklik setiap elemen init 1 posisi ke kanan;
      for setiap posisi XOR do
         init[posisi XOR + 1] ← init[posisi XOR + 1] XOR pos_trkhr;
      endfor

      gf[i, 1..m] ← init;
      gf[i, m + 1] ← i - 1;
   endfor
   gf_bin ← gf[1..2^m, 1:m];
   gf_dec ← gf[1..2^m, m + 1];
   write(gf_bin);
   write(gf_dec);
   write(m);
   write(p);
endif
```

Lampiran 2. Elemen-elemen GF(2^8) dengan primitif polinomial $p(x) = 1 + x^2 + x^3 + x^4 + x^8$

gf_bin (8-*tuple* biner)	*gf_dec* (simbol *i*)
0 0 0 0 0 0 0 0	0
1 0 0 0 0 0 0 0	1
0 1 0 0 0 0 0 0	2
0 0 1 0 0 0 0 0	3
0 0 0 1 0 0 0 0	4
0 0 0 0 1 0 0 0	5
0 0 0 0 0 1 0 0	6
0 0 0 0 0 0 1 0	7
0 0 0 0 0 0 0 1	8
1 0 1 1 1 0 0 0	9
0 1 0 1 1 1 0 0	10
0 0 1 0 1 1 1 0	11
0 0 0 1 0 1 1 1	12
1 0 1 1 0 0 1 1	13
1 1 1 0 0 0 0 1	14
1 1 0 0 1 0 0 0	15
0 1 1 0 0 1 0 0	16
0 0 1 1 0 0 1 0	17
0 0 0 1 1 0 0 1	18
1 0 1 1 0 1 0 0	19
0 1 0 1 1 0 1 0	20
0 0 1 0 1 1 0 1	21
1 0 1 0 1 1 1 0	22
0 1 0 1 0 1 1 1	23
1 0 0 1 0 0 1 1	24
1 1 1 1 0 0 0 1	25
1 1 0 0 0 0 0 0	26
0 1 1 0 0 0 0 0	27
0 0 1 1 0 0 0 0	28
0 0 0 1 1 0 0 0	29
0 0 0 0 1 1 0 0	30
0 0 0 0 0 1 1 0	31
0 0 0 0 0 0 1 1	32
1 0 1 1 1 0 0 1	33
1 1 1 0 0 1 0 0	34
0 1 1 1 0 0 1 0	35
0 0 1 1 1 0 0 1	36
1 0 1 0 0 1 0 0	37
0 1 0 1 0 0 1 0	38
0 0 1 0 1 0 0 1	39
1 0 1 0 1 1 0 0	40
0 1 0 1 0 1 1 0	41
0 0 1 0 1 0 1 1	42
1 0 1 0 1 1 0 1	43
1 1 1 0 1 1 1 0	44

gf_bin (8-*tuple* biner)	*gf_dec* (simbol *i*)
0 1 1 1 0 1 1 1	45
1 0 0 0 0 0 1 1	46
1 1 1 1 1 0 0 1	47
1 1 0 0 0 1 0 0	48
0 1 1 0 0 0 1 0	49
0 0 1 1 0 0 0 1	50
1 0 1 0 0 0 0 0	51
0 1 0 1 0 0 0 0	52
0 0 1 0 1 0 0 0	53
0 0 0 1 0 1 0 0	54
0 0 0 0 1 0 1 0	55
0 0 0 0 0 1 0 1	56
1 0 1 1 1 0 1 0	57
0 1 0 1 1 1 0 1	58
1 0 0 1 0 1 1 0	59
0 1 0 0 1 0 1 1	60
1 0 0 1 1 1 0 1	61
1 1 1 1 0 1 1 0	62
0 1 1 1 1 0 1 1	63
1 0 0 0 0 1 0 1	64
1 1 1 1 1 0 1 0	65
0 1 1 1 1 1 0 1	66
1 0 0 0 0 1 1 0	67
0 1 0 0 0 0 1 1	68
1 0 0 1 1 0 0 1	69
1 1 1 1 0 1 0 0	70
0 1 1 1 1 0 1 0	71
0 0 1 1 1 1 0 1	72
1 0 1 0 0 1 1 0	73
0 1 0 1 0 0 1 1	74
1 0 0 1 0 0 0 1	75
1 1 1 1 0 0 0 0	76
0 1 1 1 1 0 0 0	77
0 0 1 1 1 1 0 0	78
0 0 0 1 1 1 1 0	79
0 0 0 0 1 1 1 1	80
1 0 1 1 1 1 1 1	81
1 1 1 0 0 1 1 1	82
1 1 0 0 1 0 1 1	83
1 1 0 1 1 1 0 1	84
1 1 0 1 0 1 1 0	85
0 1 1 0 1 0 1 1	86
1 0 0 0 1 1 0 1	87
1 1 1 1 1 1 1 0	88
0 1 1 1 1 1 1 1	89

Lampiran 2. Lanjutan

gf_bin (8-*tuple* biner)	*gf_dec* (simbol *i*)
1 0 0 0 0 1 1 1	90
1 1 1 1 1 0 1 1	91
1 1 0 0 0 1 0 1	92
1 1 0 1 1 0 1 0	93
0 1 1 0 1 1 0 1	94
1 0 0 0 1 1 1 0	95
0 1 0 0 0 1 1 1	96
1 0 0 1 1 0 1 1	97
1 1 1 1 0 1 0 1	98
1 1 0 0 0 0 1 0	99
0 1 1 0 0 0 0 1	100
1 0 0 0 1 0 0 0	101
0 1 0 0 0 1 0 0	102
0 0 1 0 0 0 1 0	103
0 0 0 1 0 0 0 1	104
1 0 1 1 0 0 0 0	105
0 1 0 1 1 0 0 0	106
0 0 1 0 1 1 0 0	107
0 0 0 1 0 1 1 0	108
0 0 0 0 1 0 1 1	109
1 0 1 1 1 1 0 1	110
1 1 1 0 0 1 1 0	111
0 1 1 1 0 0 1 1	112
1 0 0 0 0 0 0 1	113
1 1 1 1 1 0 0 0	114
0 1 1 1 1 1 0 0	115
0 0 1 1 1 1 1 0	116
0 0 0 1 1 1 1 1	117
1 0 1 1 0 1 1 1	118
1 1 1 0 0 0 1 1	119
1 1 0 0 1 0 0 1	120
1 1 0 1 1 1 0 0	121
0 1 1 0 1 1 1 0	122
0 0 1 1 0 1 1 1	123
1 0 1 0 0 0 1 1	124
1 1 1 0 1 0 0 1	125
1 1 0 0 1 1 0 0	126
0 1 1 0 0 1 1 0	127
0 0 1 1 0 0 1 1	128
1 0 1 0 0 0 0 1	129
1 1 1 0 1 0 0 0	130
0 1 1 1 0 1 0 0	131
0 0 1 1 1 0 1 0	132
0 0 0 1 1 1 0 1	133
1 0 1 1 0 1 1 0	134

gf_bin (8-*tuple* biner)	*gf_dec* (simbol *i*)
0 1 0 1 1 0 1 1	135
1 0 0 1 0 1 0 1	136
1 1 1 1 0 0 1 0	137
0 1 1 1 1 0 0 1	138
1 0 0 0 0 1 0 0	139
0 1 0 0 0 0 1 0	140
0 0 1 0 0 0 0 1	141
1 0 1 0 1 0 0 0	142
0 1 0 1 0 1 0 0	143
0 0 1 0 1 0 1 0	144
0 0 0 1 0 1 0 1	145
1 0 1 1 0 0 1 0	146
0 1 0 1 1 0 0 1	147
1 0 0 1 0 1 0 0	148
0 1 0 0 1 0 1 0	149
0 0 1 0 0 1 0 1	150
1 0 1 0 1 0 1 0	151
0 1 0 1 0 1 0 1	152
1 0 0 1 0 0 1 0	153
0 1 0 0 1 0 0 1	154
1 0 0 1 1 1 0 0	155
0 1 0 0 1 1 1 0	156
0 0 1 0 0 1 1 1	157
1 0 1 0 1 0 1 1	158
1 1 1 0 1 1 0 1	159
1 1 0 0 1 1 1 0	160
0 1 1 0 0 1 1 1	161
1 0 0 0 1 0 1 1	162
1 1 1 1 1 1 0 1	163
1 1 0 0 0 1 1 0	164
0 1 1 0 0 0 1 1	165
1 0 0 0 1 0 0 1	166
1 1 1 1 1 1 0 0	167
0 1 1 1 1 1 1 0	168
0 0 1 1 1 1 1 1	169
1 0 1 0 0 1 1 1	170
1 1 1 0 1 0 1 1	171
1 1 0 0 1 1 0 1	172
1 1 0 1 1 1 1 0	173
0 1 1 0 1 1 1 1	174
1 0 0 0 1 1 1 1	175
1 1 1 1 1 1 1 1	176
1 1 0 0 0 1 1 1	177
1 1 0 1 1 0 1 1	178
1 1 0 1 0 1 0 1	179

Lampiran 2. Lanjutan

gf_bin (8-*tuple* biner)								*gf_dec* (simbol *i*)
1	1	0	1	0	0	1	0	180
0	1	1	0	1	0	0	1	181
1	0	0	0	1	1	0	0	182
0	1	0	0	0	1	1	0	183
0	0	1	0	0	0	1	1	184
1	0	1	0	1	0	0	1	185
1	1	1	0	1	1	0	0	186
0	1	1	1	0	1	1	0	187
0	0	1	1	1	0	1	1	188
1	0	1	0	0	1	0	1	189
1	1	1	0	1	0	1	0	190
0	1	1	1	0	1	0	1	191
1	0	0	0	0	0	1	0	192
0	1	0	0	0	0	0	1	193
1	0	0	1	1	0	0	0	194
0	1	0	0	1	1	0	0	195
0	0	1	0	0	1	1	0	196
0	0	0	1	0	0	1	1	197
1	0	1	1	0	0	0	1	198
1	1	1	0	0	0	0	0	199
0	1	1	1	0	0	0	0	200
0	0	1	1	1	0	0	0	201
0	0	0	1	1	1	0	0	202
0	0	0	0	1	1	1	0	203
0	0	0	0	0	1	1	1	204
1	0	1	1	1	0	1	1	205
1	1	1	0	0	1	0	1	206
1	1	0	0	1	0	1	0	207
0	1	1	0	0	1	0	1	208
1	0	0	0	1	0	1	0	209
0	1	0	0	0	1	0	1	210
1	0	0	1	1	0	1	0	211
0	1	0	0	1	1	0	1	212
1	0	0	1	1	1	1	0	213
0	1	0	0	1	1	1	1	214
1	0	0	1	1	1	1	1	215
1	1	1	1	0	1	1	1	216
1	1	0	0	0	0	1	1	217
1	1	0	1	1	0	0	1	218
1	1	0	1	0	1	0	0	219
0	1	1	0	1	0	1	0	220
0	0	1	1	0	1	0	1	221
1	0	1	0	0	0	1	0	222
0	1	0	1	0	0	0	1	223
1	0	0	1	0	0	0	0	224

gf_bin (8-*tuple* biner)								*gf_dec* (simbol *i*)
0	1	0	0	1	0	0	0	225
0	0	1	0	0	1	0	0	226
0	0	0	1	0	0	1	0	227
0	0	0	0	1	0	0	1	228
1	0	1	1	1	1	0	0	229
0	1	0	1	1	1	1	0	230
0	0	1	0	1	1	1	1	231
1	0	1	0	1	1	1	1	232
1	1	1	0	1	1	1	1	233
1	1	0	0	1	1	1	1	234
1	1	0	1	1	1	1	1	235
1	1	0	1	0	1	1	1	236
1	1	0	1	0	0	1	1	237
1	1	0	1	0	0	0	1	238
1	1	0	1	0	0	0	0	239
0	1	1	0	1	0	0	0	240
0	0	1	1	0	1	0	0	241
0	0	0	1	1	0	1	0	242
0	0	0	0	1	1	0	1	243
1	0	1	1	1	1	1	0	244
0	1	0	1	1	1	1	1	245
1	0	0	1	0	1	1	1	246
1	1	1	1	0	0	1	1	247
1	1	0	0	0	0	0	1	248
1	1	0	1	1	0	0	0	249
0	1	1	0	1	1	0	0	250
0	0	1	1	0	1	1	0	251
0	0	0	1	1	0	1	1	252
1	0	1	1	0	1	0	1	253
1	1	1	0	0	0	1	0	254
0	1	1	1	0	0	0	1	255

Lampiran 3. Algoritma GF2_Kali

Algoritma GF2_Kali

'Algoritma untuk melakukan operasi perkalian pada GF(2^m)'

```
DEKLARASI 'nilai m, gf_dec, gf_bin didapatkan dari fungsi GF2'
global m      : integer 'derajat polinomial primitif p(x)'
global gf_dec: array [1..2^m,1]of integer 'representasi elemen  simbol i, 0 ≤
              i  ≤ 2^m - 1'
global gf_bin: array  [1..2^m,1..m]of  integer  'representasi  elemen  m-tuple
              biner'
a,b           : integer 'a dan b representasi  elemen simbol i, 0 ≤ i ≤ 2^m - 1'
kali          : integer 'hasil perkalian'

DESKRIPSI
read(a,b);
pastikan a dan b selalu positif;
if a dan b bukan elemen pada gf_dec then
   program berhenti dengan pesan kesalahan;
else
   if a = 0 or b = 0 then
      kali ← 0;
   else
      'setiap  input  a,b  dicari  representasi  pangkat  α,  kemudian  diproses,
      setelah proses selesai, hasilnya akan diubah ke representasi simbol i . '
      kali ← (((gf_dec[a+1]-1) + (gf_dec[b+1]-1)) mod 2^m-1) + 1;
   endif
   write(kali);
endif
```

Lampiran 4. GF2 _Tambah

Algoritma GF2_Tambah

'Algoritma untuk melakukan operasi penjumlahan pada GF(2^m)'

```
DEKLARASI 'nilai m, gf_dec, gf_bin didapatkan dari fungsi GF2'
global m      : integer 'derajat polinomial primitif p(x)'
global gf_dec: array [1..2^m,1]of integer 'representasi elemen  simbol i, 0 ≤
              i ≤ 2^m - 1'
global gf_bin: array  [1..2^m,1..m]of  integer  'representasi  elemen  m-tuple
              biner'
a,b           : integer 'a dan b representasi  elemen simbol i, 0 ≤ i ≤ 2^m - 1'
tambah        : integer 'hasil penjumlahan'

DESKRIPSI
read(a,b);pastikan a dan b selalu positif;
if a dan b bukan elemen pada gf_dec then
   program berhenti dengan pesan kesalahan;
else
   tambah = gf_bin[a+1,1..m] XOR gf_bin[b+1,1..m];'operasi penjumlahan'
   for i ←1 to 2^m do
      if tambah = gf_bin[i,1..m] then
         tambah = gf_dec[i]; 'dapatkan representasi simbol i'
         break; 'proses pencarian dihentikan'
      endif
   endfor
   write(tambah);
endif
```

Lampiran 5. Algoritma GF2 _Invers

Algoritma GF2_Invers
'Algoritma untuk melakukan operasi invers pada GF(2^m)'

```
DEKLARASI 'nilai m, gf_dec, gf_bin didapatkan dari fungsi GF2'
global m      : integer 'derajat polinomial primitif p(x)'
global gf_dec: array [1..2^m,1]of integer 'representasi elemen  simbol i, 0 ≤
              i ≤ 2^m - 1'
global gf_bin: array  [1..2^m,1..m]of  integer  'representasi  elemen  m-tuple
              biner'
a             : integer 'a  representasi  elemen simbol i, 0 ≤ i ≤ 2^m - 1'
invers        : integer 'hasil invers'

DESKRIPSI
read(a);pastikan a selalu positif;
if a bukan elemen pada gf_dec then
   program berhenti dengan pesan kesalahan;
else
   if a = 0 then
      invers ← Inf; 'infinity'
   else
      'setiap input a dicari representasi pangkat α, diproses (2^m - 1) - a,
      setelah proses selesai, hasilnya akan diubah ke representasi simbol i .'
      invers ← (((2^m - 1) - (gf_dec(a+1)-1)) mod 2^m-1) + 1;
   endif
   write(invers);
endif
```

Lampiran 6. GF2 _Bagi

Algoritma GF2_Bagi
'Algoritma untuk melakukan operasi pembagian pada GF(2^m)'

```
DEKLARASI 'nilai m, gf_dec, gf_bin didapatkan dari fungsi GF2'
global m      : integer 'derajat polinomial primitif p(x)'
global gf_dec: array [1..2^m,1]of integer 'representasi elemen  simbol i, 0 ≤
              i ≤ 2^m - 1'
global gf_bin:  array  [1..2^m,1..m]of  integer  'representasi  elemen  m-tuple
              biner'
a,b           : integer 'a dan b representasi  elemen simbol i, 0 ≤ i ≤ 2^m - 1'
bagi          : integer 'hasil pembagian'

DESKRIPSI
read(a,b);pastikan a dan b selalu positif;
if a dan b bukan elemen pada gf_dec then
   program berhenti dengan pesan kesalahan;
else
   if a ≥ 0 and b = 0 then bagi ← Inf; 'infinity';
   else
      if a = 0 and b > 0 then bagi ← 0;
      else
         bagi ← GF2_Kali(a, GF2_Invers(b));
      endif
   endif
   write(bagi);
endif
```

Lampiran 7. Algoritma Carids

Algoritma Carids

'Algoritma untuk mencari panjang kode yang sesuai. Untuk input panjang kode genap akan dicari panjang kode ganjil yang sesuai'

```
DEKLARASI
n        : integer 'panjang kode ganjil atau genap'
d        : integer 'panjang kode ganjil'
s        : integer 's merupakan 2^j, j ≥ 1'
pngkt2   : integer 'variabel untuk menghitung s'

DESKRIPSI
read(n); 'pastikan n  selalu integer positif > 0'
if n bukan integer positif > 0 then
   program berhenti dengan pesan kesalahan;
else
   if n genap then
      pngkt2 ← 2; 'inisialisasi'
      d ← 0; 'inisialisasi'
      while d masih genap do
         'berdasarkan persamaan Lemma1 maka (x^n + 1) = (x^d + 1)^s'
         d ← n / pngkt2;
         s ← pngkt2;
              pngkt2 ← 2 * pngkt2;
      endwhile
      n ← d; 'panjang n ganjil'
   else
      s ← 1;
   endif
   write(n); 'n ganjil'
   write(s);
endif
```

Lampiran 8. Algoritma Carim

Algoritma Carim

'Algoritma untuk mencari m multiplicative order 2 mod n yang merupakan derajat polinomial primitif p(x)'

```
DEKLARASI
n  : integer 'panjang kode ganjil'
m  : integer 'multiplicative order 2 mod n, derajat polinomial primitif p(x)'

DESKRIPSI
read(n);
if n bukan integer ganjil positif > 0 then
   program berhenti dengan pesan kesalahan;
else
   m ← 1; 'inisialisasi'
   while (2^m - 1 mod n) ≠0 do 'Mencari m terkecil sedemikian sehingga n|2^m-1'
      m ← m + 1;
   endwhile
   write(m);
endif
```

Lampiran 9. Algoritma Conjugate

Algoritma Conjugate

'Algoritma untuk mencari conjugate suatu akar pangkat α'

```
DEKLARASI
s                : integer 'representatif koset mod n'
n                : integer 'panjang kode, bilangan bulat ganjil positif > 0'
konj             : array[] of integer 'matriks baris representasi s dan
                 conjugatenya'
hitplus, hitung : integer 'variabel penghitung conjugate '
e                : integer 'bilangan integer postitif terkecil sehingga 2^e s ≡ s'

DESKRIPSI
read(s,n);
if n bukan integer ganjil positif > 0 or s < 0 or s ≥ n then
   program berhenti dengan pesan kesalahan;
else
   konj ← s; hitplus ← s; hitung ← 0; e ← 0; 'inisialisasi'
   while hitung ≠ s do
      'proses dilakukan berdasarkan pembentukan koset siklotomik mod n atas
   GF(2)'
      e ← e+1;
      hitung ← (2 * hitplus) mod n;
      if hitung = s then
         looping dihentikan;
      endif
      konj ← gabungkan  konj sebelumnya dengan hitung secara horizontal;
      hitplus ← hitung;
   endwhile
   write(konj);
   write(e);
endif
```

Lampiran 10. Algoritma Cyclotomic

Algoritma Cyclotomic

'Algoritma untuk menciptakan semua koset siklotomik dari suatu kode dengan panjang n ganjil'

```
DEKLARASI
n     : integer
cs    : array[] of integer 'representasi koset-koset siklotomik'
s     : array[] of integer 'matriks baris yang berisi representatif koset mod
      n'
c     : array[1..n-1] of integer 'representasi gabungan elemen-elemen koset
      kecuali 0'
caris : array[] of integer
hasil : array[] of integer 'matriks berisi s dan conjugatenya'

procedure Conjugate(input s_cs, n : integer)

DESKRIPSI
read(n);
if n bukan bilangan integer ganjil positif > 0 then
   program berhenti dengan pesan kesalahan;
else
   if n = 1
      c ← [];
```

Lampiran 10. Lanjutan

```
  else
      c ← [1, 2, 3, ..., n-1]; 'gabungan elemen-elemen koset siklotomik
                                kecuali 0'
  end

  caris ← c; 'inisialisasi'
  cs ← [0];'inisialisasi'
  s ← [0]; 'inisialisasi'
  while masih ada elemen caris do
     'cari[1] adalah coset representative, elemen terkecil untuk setiap koset'
     hasil ← conjugate(caris[1],n);
     s ← gabungkan s dan cari[1] secara horizontal;
     cs ← gabungkan cs dan hasil secara vertikal;
     caris ← operasi pengurangan himpunan pada matriks caris dengan hasil;
  endwhile
  write(cs);
  write(s);
endif
```

Lampiran 11. Hasil algoritma Cyclotomic untuk $n = 255$

cs =

0	0	0	0	0	0	0	0
1	2	4	8	16	32	64	128
3	6	12	24	48	96	192	129
5	10	20	40	80	160	65	130
7	14	28	56	112	224	193	131
9	18	36	72	144	33	66	132
11	22	44	88	176	097	194	133
13	26	52	104	208	161	67	134
15	30	60	120	240	225	195	135
17	34	68	136	0	0	0	0
19	38	76	152	49	98	196	137
21	42	84	168	081	162	69	138
23	46	92	184	113	226	197	139
25	50	100	200	145	35	70	140
27	54	108	216	177	99	198	141
29	58	116	232	209	163	71	142
31	62	124	248	241	227	199	143
37	74	148	41	82	164	73	146
39	78	156	57	114	228	201	147
43	86	172	89	178	101	202	149
45	90	180	105	210	165	75	150
47	94	188	121	242	229	203	151
51	102	204	153	0	0	0	0
53	106	212	169	830	166	77	154
55	110	220	185	115	230	205	155
59	118	236	217	179	103	206	157
61	122	244	233	211	167	79	158
63	126	252	249	243	231	207	159
85	170	0	0	0	0	0	0
87	174	93	186	117	234	213	171
91	182	109	218	181	107	214	173
95	190	125	250	245	235	215	175
111	222	189	123	246	237	219	183
119	238	221	187	0	0	0	0
127	254	253	251	247	239	223	191

s = 0 1 3 5 7 9 11 15 15 17 19 21 23 25 27 29 31 37 39 43 45
47 51 53 55 59 61 63 85 87 91 95 111 119 127

Lampiran 12. Algoritma KodeSiklik

Algoritma KodeSiklik

'Algoritma untuk menciptakan kode siklik dengan panjang kode n'

```
DEKLARASI
n    : integer 'panjang kode siklik genap atau ganjil'
d    : integer 'panjang kode siklik ganjil'
s    : integer 'apabila n genap maka s genap'
cs   : array[] of integer 'koset-koset siklotomik'
s_cs : array[] of integer 'matriks baris yang berisi representatif koset
     minimum mod n setiap koset siklotomik'
m    : integer 'derajat polinomial primitif p(x)'
r    : integer 'pangkat α pembuat siklik'
p    : array[] of integer 'matriks koefisien polinomial primitif p(x)
     ascending power'
gf   : array[,] of integer 'representasi elemen GF(2^m) dalam bentuk m-tuple
     biner'

procedure Carids(input n : integer)
procedure Cyclotomic(input d : integer)
procedure Carim(input d : integer)
procedure GF2(input p : array [] of integer)

DESKRIPSI
read(n);
if n bukan bilangan integer positif > 0 then
   program berhenti dengan pesan kesalahan;
else
   'Mencari panjang kode siklik'
   [d,s] ← Carids(n);
   write(n);
   write(d);
   write(s);

   'Menciptakan koset siklotomik.'
   [cs,s_cs] ← Cyclotomic(d);
   write(cs);
   write(s_cs);

   'Mencari nilai m'
   m ← Carim(d);
   write(m);

   'Mencari pangkat α pembuat siklik'
   r ← (2^m - 1) / d; 'berdasarkan persamaan (α^r)^n = (α)^(2^m - 1) sehingga r=(2^m-1)/n'
   write(r);

   'Menciptakan GF(2^m)'
   gf, p ← GF2(m);
   write(gf);
   write(p);
endif
```

Lampiran 13. Hasil algoritma KodeSiklik semua kemungkinan katakode untuk panjang kode $n = 31$

n = 31 (panjang kode *input*)
d = 31 (panjang kode ganjil)
s = 1 (pangkat dari $x^d + 1$)
cs = 0 0 0 0 0 (koset-koset siklotomik)
1 2 4 8 16
3 6 12 24 17
5 10 20 9 18
7 14 28 25 19
11 22 13 26 21
15 30 29 27 23
s_cs = 0 1 3 5 7 11 15 (representatif mod 31)
m = 5 (nilai *m*)
r = 1 (pangkat α pembuat siklik)
p = 1 0 1 0 0 1 ($p(x) = 1 + x^2 + x^5$)
gf = (elemen 5-*tuple* biner $GF(2^5)$)

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

Lampiran 14. Algoritma MinPoly

Algoritma MinPoly

'Algoritma untuk mencari minimal polinomial dari suatu koset menggunakan suatu representatif koset s'

DEKLARASI

```
root     : integer 'representatif koset s'
n        : integer 'panjang kode siklik ganjil'
input    : array[] of integer
a,b,w    : array[] of integer 'representasi koefisien polinomial descending
           power'
r        : integer 'pangkat α pembuat siklik'
i,j,k    : integer 'indeks'
n,o      : integer 'variabel penghitung banyaknya kolom'
minpoly  : array[] of integer 'representasi koefisien polinomial minimal
           ascending power'
conjgt   : array[] of integer 'representasi root dan conjugatenya'

procedure GF2_tambah(input x,y : integer)
procedure GF2_kali(input x,y : integer)
procedure Carim(input n : integer)
procedure Conjugate(input root, n : integer)

DESKRIPSI
read(root,n);
if root ≥ n or n < 1 or root < 0 then
  program berhenti dengan pesan kesalahan;
else
  if root = 0 and n>0 then
    minpoly ← [1 1];
  else
    m ← Carim(n);
    r ← (2^m - 1) / n;
    input ← Conjugate(root,n); 'mencari conjugate'
    cnjgt ← input;
    input ← input * r;
    w ← []; 'inisialisasi'
    cari banyaknya kolom matriks input;

    for i←1 to banyaknya kolom matriks input - 1 do
      a ← w;
        b ←[1 input[i+1]+1];

      if i = 1 then
        a ← [1 input[1]+1];'representasi x + α^j, j adalah elemen pangkat α'
        b ← [1 input[2]+1];
      endif

      n ← banyaknya kolom matriks a;
      o ← banyaknya kolom matriks b;
      k ← o + n - 1;
      w[1..k] ← [0, 0 ,0 , ..., 0];'inisialisasi'

      'proses perkalian'
      for i ← 1 to k
            for j ← cari nilai maksimum (1, i + 1 - m) to cari nilai
                minimum (i, n) do
```

Lampiran 14. Lanjutan

```
                'matriks w merupakan koefisien polinomial dengan descending
                power'
                w[1,i] ← GF2_tambah(w[1,i], GF2_kali(a[j],b[i + 1 - j]));
            endfor
        endfor
    endfor
    w ← balik elemen-elemen w;'representasi koefisien polinomial minimal
                                ascending power'
    minpoly ← w;
    write(minpoly);
    write(cnjgt);
  endif
endif
```

Lampiran 15. Algoritma GF2_TampilPoly

Algoritma GF2_TampilPoly

'Algoritma untuk menampilkan polinomial dalam bentuk polinomial x dengan koefisien atas GF(2^m)'

```
DEKLARASI
input: array[] of integer 'representasi koefisien polinomial (elemen-elemen
    GF(2^m) dalam bentuk simbol i, apabila input=1 maka input=α^0 =1 )'
a    : array[] of string 'variabel penghitung polinomial x'
cari : array[] of integer 'variabel pencari elemen taknol'
i    : integer 'indeks'

DESKRIPSI
read(input);
if input bukan bilangan integer positif then
  program berhenti dengan pesan kesalahan;
else
  cari ← cari elemen matriks input ≠ 0;
  if banyaknya kolom cari = 0 then 'input berupa string kosong'
    a ← [];
  else
    if input[1] = 1 then
      a ← ['1'];
    else
      if input[1] > 1 then
        a ← ['α^' 'input[1]-1'];
      else
        if cari[1] = 2 then
          if input[cari[1]] = 1 then
            a ← ['X']; 'apabila elemen ke-2 input = 1'
          else
            a ← ['α^' 'input[cari[1]] - 1' ' X'];
          endif
        else
          if cari[1]>2 then
            if input[cari[1]] = 1
              a ← ['X^' 'cari[1]-1'];'apabila elemen ke-i input = 1, i>2'
            else
              a ← ['α^' 'input[cari[1]]-1' ' X^' 'cari(1)-1)'];
            endif
          endif
        endif
```

Lampiran 15. Lanjutan

```
        endif
      endif
    endif
    for i ←2 to banyaknya kolom cari do
      if input[cari[i]] = 1 and cari[i] = 2
        a ← [a ' + ' 'X'];
      else
        if input[cari[i]] > 1 and cari[i] = 2
          a ← [a ' + α^' 'input[cari[i]]-1' ' X'];
        else
          if input(cari(i)) == 1
            a ← [a ' + ' 'X^' 'cari[i]-1'];
          else
            a ← [a ' + α^' 'input[cari[i]]-1' ' X^' 'cari[i]-1'];
          endif
        endif
      endif
    endfor
    write(a);'representasi polinomial x atas GF(2^m)'
endif
```

Lampiran 16. Algoritma Reciprocal

Algoritma Reciprocal

'Algoritma untuk mencari polinomial reciprocal dari suatu polinomial'

```
DEKLARASI
input    : array[] of real 'representasi koefisien polinomial dalam x ascending
           power'
rcprcl   : array[] of real 'representasi koefisien polinomial reciprocal dalam
           x ascending power'

DESKRIPSI
read(input);
if setiap elemen input bukan bilangan nyata then
   program berhenti dengan pesan kesalahan;
else
   rcprcl ← balik posisi input;
   write(rcprcl);
endif
```

Lampiran 17. Algoritma Matriks_MinPoly_XR

Algoritma Matriks_MinPoly_XR

'Algoritma untuk mencari polinomial minimal dalam x menggunakan polinimial reciprocal'

```
DEKLARASI
s_cs     : array[]  of  integer  'matriks  baris  yang  setiap  elemennya
           merepresentasikan s pada M^(s)(x)'
n        : integer 'panjang kode'
mp       : array[,] of string 'matriks yang setiap barisnya merepresentasikan
           s, M^(s)(x)'
mp1      : array[] of integer 'variabel pencari polinomial minimal'
mp2      : array[] of string 'variabel pencari polinomial minimal dalam x'
x        : integer 'representatif koset minimum mod n milik polinomial minimal
           reciprocal'
```

Lampiran 17. Lanjutan

```
maks     : integer 'representatif koset maksimum mod n pada sebuah koset'
cnjgt    : array[] of integer 'representasi  sebuah koset'
s_cnjgt  : integer 'representatif koset minimum mod n pada sebuah koset'

procedure GF2_tampilpoly(input a : array[] of integer)
procedure Minpoly(input a : integer, b : integer)
procedure Conjugate(input root, n : integer)

DESKRIPSI
read(s_cs,n);
if s_cs,n bukan bilangan integer positif or n<1 or setiap elemen s_cs < 0 or
setiap elemen s_cs ≥ n then
   program berhenti dengan pesan kesalahan;
else
   mp←[];'inisialisasi'
   while masih ada elemen s_cs do
      cnjgt ← Conjugate(s_cs[1],n); 'mencari conjugate'
      if s_cs[1] = 0 then
         mp2 ← [' 0, 1 + X '] ; 'representasi C0'
         mp ← gabungkan mp dan mp2 secara vertikal;
         s_cs ← operasi pengurangan himpunan s_cs dengan 0;
      else 'apabila terdapat kasus khusus maka x^n+ 1 = (x+1) (x^n-1 + ... + x+1)'
         if terjadi kasus khusus then
            mp ← [' 0, 1 + X '];
            mp2←[' 1, 1 + X + X^2 + ... + X^n - 1'];
            mp← gabungkan mp dan mp2 secara vertikal;
            s_cs ← operasi pengurangan himpunan s_cs dengan 0 dan cnjgt;
         else
            mp1 ← Minpoly(s_cs[1],n);
            s_cnjgt ← cari representatif koset minimum mod n dari cnjgt;
            mp2 ←['s_cs[1], GF2_tampilpoly(mp1)']; 'representasi s, M(s)(x)'
            s_cs ← operasi pengurangan himpunan s_cs dengan cnjgt;
            mp ← gabungkan mp dan mp2 secara vertikal;

            'mencari polinomial reciprocal masing-masing koset siklotomik'
            maks ← max(cnjgt); 'representatif koset maksimum mod n untuk setiap
                                conjugate'
            x ← n - maks;
            if poliomial minimal memiliki polinomial reciprocal bukan dirinya
               sendiri then
               mp1 ← reciprocal(mp1);
               mp2 ←['x, gf2_tampilpoly(mp1)];
               mp ← gabungkan mp dan mp2 secara vertikal;
               cnjgt ← cari conjugate dari x;
               s_cs ← operasi pengurangan himpunan s_cs dengan cnjgt;
            endif
         endif
      endif
   endwhile
   write(mp);
endif
```

Lampiran 18. Algoritma Matriks_MinPoly_X

Algoritma Matriks_MinPoly_X
'*Algoritma untuk mencari polinomial minimal dalam x*'

```
DEKLARASI
s_cs     : array[]  of  integer  'matriks  baris  yang  setiap  elemennya
           merepresentasikan s pada M(s) (x)'
n        : integer 'panjang kode'
mp       : array[,] of string 'matriks yang setiap barisnya merepresentasikan
           s, M(s) (x)'
mp1      : array[] of string 'variabel pencari polinomial minimal dalam x'
min_poly : array[] of integer 'variabel pencari polinomial minimal'
cnjgt    : array[] of integer 'representasi  sebuah koset'

procedure GF2_tampilpoly(input a : array[] of integer)
procedure Minpoly(input a : integer, b : integer)
procedure Conjugate(input root, n : integer)

DESKRIPSI
read(s_cs,n);
if s_cs,n bukan bilangan integer positif or n<1 or setiap elemen  s_cs < 0 or
setiap elemen s_cs ≥ n then
   program berhenti dengan pesan kesalahan;
else
   mp ← [];'inisialisasi'
   while masih ada elemen s_cs do
        cnjgt ← conjugate(s_cs[1],n); 'mencari conjugate'
        if s_cs[1] = 0 then
          mp1 ← [' 0, 1 + X '] ; 'representasi C0'
          mp ← gabungkan mp dan mp2 secara vertikal;
          s_cs ← operasi pengurangan himpunan s_cs dengan 0;
        else
          if terjadi kasus khusus then
               mp ← [' 0, 1 + X ']; mp1 ← [' 1, 1 + X + X^2 + ... + X^n - 1'];
               mp ← gabungkan mp dan mp1 secara vertikal;
               s_cs ← operasi pengurangan himpunan s_cs dengan 0 dan cnjgt;
          else
               min_poly ← minpoly(s_cs[1],n);
               mp1 ← ['s_cs[1], gf2_tampilpoly(min_poly)']; 'representasi s,
                                                          M(s) (x)'
               s_cs ← operasi pengurangan himpunan s_cs dengan cnjgt;
               mp ← gabungkan mp dan mp1 secara vertikal;
        endif
     endif
   endwhile
   write(mp);
endif
```

Lampiran 19. Algoritma Eksp

Algoritma Eksp
'*algoritma untuk mencari designed distance, b, dan indeks b, b+1, b+2, ..., b+δ-2*'

```
DEKLARASI
gbngn    : array[] of integer 'matriks baris representasi gabungan elemen-
           elemen koset'
b        : integer 'representasi b'
```

Lampiran 19. Lanjutan

```
bd      : array[] of integer 'matriks baris representasi gabungan b apabila
          terdapat designed distance maksimum yang sama'
b1      : integer 'variabel penyimpan sementara'
cari_s_g: array[] of integer 'variabel penghitung  b,b+1,b+2, ...,b+δ -2'
s_g     : array[,] of integer 'matriks representasi b,b+1,b+2, ...,b+δ -2 yang
          memiliki designe distance sama'
dd      : integer 'designed distance'
ddl     : array[] of integer 'matrie baris representasi designed distance yang
          lainnya'
d       : integer 'variabel pencari designed distance'
i       : integer 'indeks'

DESKRIPSI
read(gbngn);
if  gbngn bukan bilangan integer positif or setiap elemen gbngn < 0 then
   program berhenti dengan pesan kesalahan;
else
   if gbngn hanya memiliki 1 elemen then
      b1 ← gbngn[1]; bd ← b1; b ← b1; cari_s_g←b1; s_g ← b1;
      dd ← gbngn[1] + 2 - b; ddl ← [];
   else
      gbngn ← pastikan elemen gbngn terurut dan unik;
      b1 ← gbngn[1];'inisialisasi'
      bd ← [];'inisialisasi'
      b ← b1;'inisialisasi'
      cari_s_g ← b1; 'inisialisasi'
      s_g ← [];'inisialisasi'
      dd ← 0; 'inisialisasi'
      ddl ← [];'inisialisai'
      d←gbngn[1] + 2 - b;'inisialisasi'

      for i ←2 to banyaknya kolom gbngn do
         b1 ← b1+1;
         if gbngn[i] = b1 then
            cari_s_g ← gabungkan cari_s_g dan gbngn[i] secara horizontal;
                                                'membentuk urutan b, b+1,
                                                b+2, ...,b+δ -2'
            d ← gbngn[i] + 2 - b;
         else
            if d > dd then
               dd ← d;
               bd ← b;
               b ← gbngn[i]; 'inisialisasi ulang'
               s_g ← cari_s_g;
            else
               if d = dd then
                  bd ← gabungkan bd dan b secara horizontal;
                  b ← gbngn[i]; 'inisialisasi ulang'
                  s_g ← gabungkan s_g dan cari_s_g secara vertikal;
               else
                  b ← gbngn[i]; 'inisialisasi ulang'
               endif
            endif

            'inisialisasi ulang'
            b1 ← gbngn[i];
```

Lampiran 19. Lanjutan

```
            ddl ← gabungkan ddl dan d secara vertikal;
            d ← gbngn[i]+2-b;
            cari_s_g ← gbngn[i];
            '--------------------'
         endif

         if i = banyaknya kolom gbngn then
            if d > dd
               dd ← d;
               bd ← b;
               s_g ← cari_s_g;
            else
               if d = dd then
                  bd ← gabungkan bd dan b secara horizontal;
                  s_g ← gabungkan s_g dan cari_s_g secara vertikal;
               endif
            endif

            ddl ← gabungkan ddl dan d secara horizontal;
         endif
      endfor
      write(dd);
      write(s_g);
      write(bd);
      write(ddl);
   endif
endif
```

Lampiran 20. Algoritma Minpoly_Generator

Algoritma Minpoly_Generator

'Algoritma untuk mencari indeks-indeks polinomial generator'

```
DEKLARASI
s_cs          : array[,] of integer 'matriks representasi g(x) = LCM{M^(b)(x),
                M^(b+1)(x), ..., M^(b + δ -2)(x)}'
n             : integer 'panjang kode'
s_mp_zeros    : array[,] of integer 'matriks representasi indeks-indeks
                polinomial generator'
s_mp_zeros1   : array[] of integer 'variabel penyimpan sementara'
der_g         : array[,] of integer 'matriks kolom representasi derajat
                polinomial generator'
cari_der_g    : integer 'derajat polinomial hasil perkalian polinomial-
                polinomial minimal'
der_g1        : integer 'derajat polinomial minimal'
i             : integer 'indeks'
cari_s_cs     : array[] of integer 'representasi setiap baris s_cs apabila
                input s_cs lebih dari 1 baris'
cnjgt         : array[] of integer 'representasi sebuah koset'
s_cnjgt       : integer 'representatif koset minimum mod n pada sebuah coset'

procedure Conjugate(input root, n : integer)

DESKRIPSI
if s_cs,n bukan bilangan integer positif or n<1 or s_cs < 0 or s_cs ≥ n then
   program berhenti dengan pesan kesalahan;
else
```

Lampiran 20. Lanjutan

```
   for i ←1 to banyaknya baris s_cs do
      cari_s_cs ← pastikan elemen setiap baris s_cs terurut dan unik;
      s_mp_zeros1 ← [];'inisialisasi'
      while masih ada elemen cari_s_cs do
         [cnjgt,der_g1] ← conjugate(cari_s_cs[1],n);
         cari_der_g  ← cari_der_g + der_g1;
         s_cnjgt ← cari representatif koset minimum mod n untuk cnjgt;
         s_mp_zeros1 ← gabungkan s_mp_zeros1 dan s_cnjgt secara horizontal;
         cari_s_cs ← operasi pengurangan himpunan cari_s_cs dengan cnjgt;
      endwhile
      der_g  ← gabungkan der_g dan cari_der_g secara vertikal;

      if i ← 1 then
         s_mp_zeros ← s_mp_zeros1;
      else
         s_mp_zeros ← gabungkan s_mp_zeros dan s_mp_zeros1 secara vertikal;
      endif
   endfor
   write(s_mp_zeros);
   write(der_g);
endif
```

Lampiran 21. Algoritma Polinomial_Generator

Algoritma Polinomial_Generator
'Algoritma untuk mencari polinomial generator'

```
DEKLARASI
g       : string 'polinomial generator g(x) atas GF(2)'
koef_g  : array[] of integer 'koefisien g(x) ascending power'

DESKRIPSI
read(polinomial-polinomial minimal);
g ← LCM dari polinomial-polinomial minimal;
koef_g ← dapatkan koefisien g(x) dengan ascending power;
write(koef_g);
```

Lampiran 22. Algoritma BCH_List

Algoritma BCH_List
'algoritma untuk menentukan designed distance, indeks polinomial generator dan dimensi kode BCH narrow sense'

```
DEKLARASI
cs          : array[,] of integer 'representasi koset-koset siklotomik'
n           : integer 'panjang kode BCH'
s_mp        : array[,] of integer 'matriks representasi indeks polinomial
              generator, khusus untuk s_mp[1,1] merepresentasikan polinomial
              generator g(x) = 1 '
s_mp1       : array[] of integer 'variabel pencari indeks polinomial generator '
dim         : array[,] of integer 'matriks kolom representasi dimensi kode BCH
              dengan polinomial generator tertentu'
dimension   : integer 'variabel penghitung dimensi'
jd          : integer 'jarak desain'
bd          : array[] of integer 'representasi b, b+1, b+2, ...,b+δ-2'
```

Lampiran 22. Lanjutan

```
gbngn      : array[] of integer 'matriks baris representasi gabungan elemen-
           elemen koset'
i,j        : integer 'indeks'

procedure Carids(input n : integer)
procedure Cyclotomic(input n: integer)
procedure Eksp(input gbngn : array[] of integer)
procedure Minpoly_Generator(input bd : array of integer, n : integer)

DESKRIPSI
read(n);
if n bukan bilangan integer positif or n<1 then
   program berhenti dengan pesan kesalahan;
else
   n ← Carids(n); 'mencari n ganjil'
   cs ← Cyclotomic(n);
   '---inisialisasi---'
   s_mp ← [1];
   s_mp1 ← [];
   dim ← [n];
   dd ← [1];
   j←2;
   gbngn←[];
   '----------------'
   for i←2 to banyaknya kolom cs do
      'Mencari designed distance'
      cari_dd ← [];
      s_mp1 ← [];
      gbngn ← gabungkan elemen koset dengan elemen koset sebelumnya;
      [jd,bd] ← Eksp(gbngn);
      while 2*j-1 ≤ jd do
         'ketika mencari dd=jd pada kode BCH ternyata didapatkan dd ganjil yang
         lain'
         cari_dd ← gabungkan cari_dd dengan 2*j-1 secara horizontal;
         j ← j + 1;
      endwhile

      dd ← gabungkan dd dengan cari_dd secara vertikal;
      '--------------------------------'
      'Mencari indeks polinomial generator'
      [indeks,der_g]  ←  Minpoly_Generator(bd[1,1...,banyaknya  kolom  bd],n);
                                                    'untuk b = 1'
      s_mp1 ← gabungkan s_mp1 sebelumnya dengan indeks secara horizontal;
      s_mp ← gabungkan s_mp sebelumnya dengan s_mp1 secara vertikal;
      '------------------------------'
      'Mencari Dimensi Kode BCH'
      dimension ← n - der_g;
      dim ← gabungkan dim sebelumnya dengan dimension secara vertikal;
      '----------------------------'
   endfor
   list ← gabungkan dd, s_mp dan dim secara horizontal;

   write(dd);
   write(s_mp);
   write(dim);
   write(list);
endif
```

Lampiran 23. Algoritma LFSR

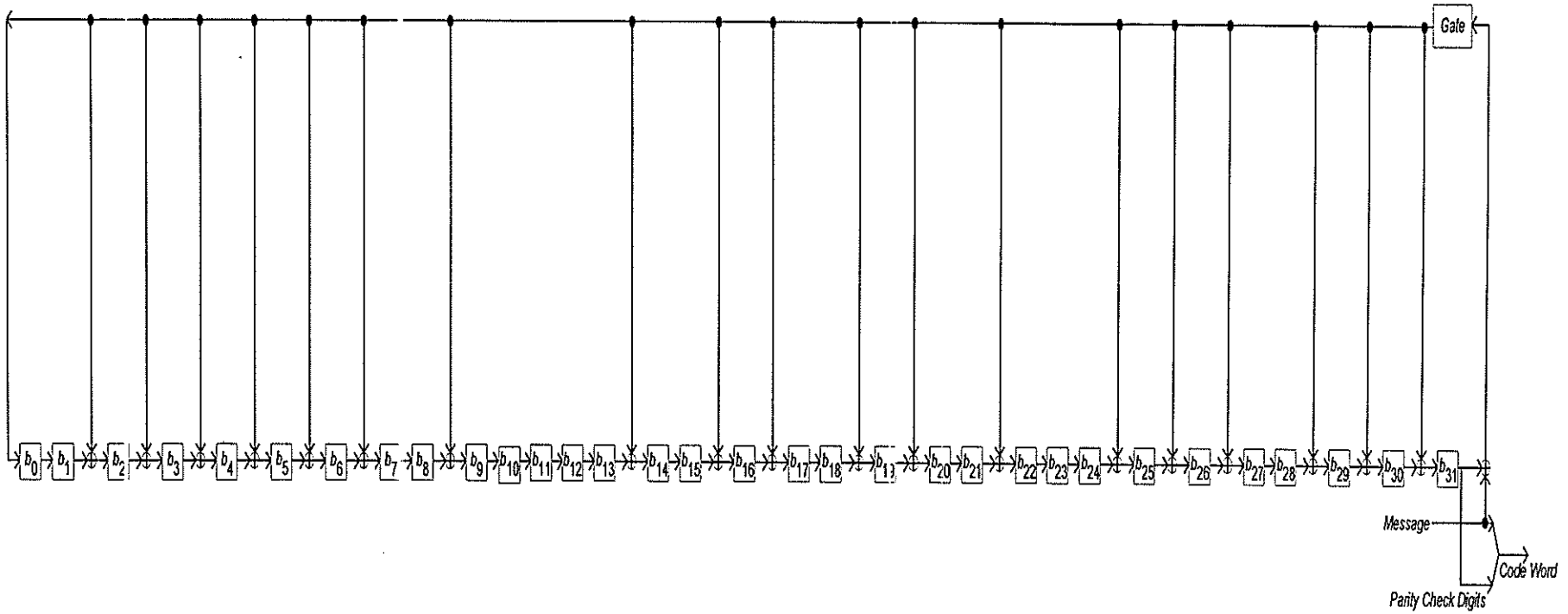
Algoritma LFSR

'Algoritma untuk mencari posisi XOR pada register geser umpan balik'

```
DEKLARASI
g       : array[] of integer 'representasi koefisien polinomial generator g(x)
          atas GF(2) ascending power'
PXOR    : array[] of integer 'representasi posisi XOR pada LFSR'
der_g   : integer 'derajat polinomial g(x)'

DESKRIPSI
read(g);
if  g bukan polinomial dengan koefisien atas GF(2) then
   program berhenti dengan pesan kesalahan;
else
   g ← pastikan g adalah representasi koefisien polinomial monic unik g(x);
   der_g ← banyaknya kolom g - 1
   PXOR ← dapatkan posisi xor;
   write(der_g);
   write(PXOR);
endif
```

Lampiran 24. Bentuk sirkuit *encoding* dari kode BCH dengan polinomial generator $g(x) = 1 + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^9 + x^{14} + x^{16} + x^{17} + x^{19} + x^{20} + x^{22} + x^{25} + x^{27} + x^{26} + x^{29} + x^{31} + x^{30} + x^{32}$ dan panjang kode $n = 255$.



Lampiran 25. Algoritma Bobot

Algoritma Bobot
'Algoritma untuk mencari bobot suatu katakode'

```
DEKLARASI
katakode   : array[] of integer 'representasi katakode atas GF(2)'
bbt        : integer 'bobot katakode'
DESKRIPSI
read(katakode);
if  katakode bukan katakode atas GF(2) then
   program berhenti dengan pesan kesalahan;
else
   bbt ← banyaknya elemen taknol dari katakode;
   write(bbt);
endif
```

Lampiran 26. Encoding_BCH

Algoritma Encoding_BCH
'Algoritma untuk encoding kode BCH'

```
DEKLARASI
psn            :  array[,] of integer 'pesan biner'
n              :  integer 'panjang kode BCH'
g              :  array[] of integer 'koefisien g(x) atas GF2 ascending power'
katakode       :  array[,] of integer
jarak_minimum  :  integer
bbt            :  integer 'bobot setiap katakode'
i,j,k          :  integer 'indeks'
init           :  array[] of integer 'representasi elemen LFSR'
PXOR           :  array[] of integer 'posisi XOR  LFSR'
pos_trkhr      :  integer 'hasil operasi XOR  setiap elemen pesan dengan elemen
                  terakhir LFSR'

procedure LFSR(input g : array[] of integer)
procedure Bobot(input katakode : array[,] of integer)

DESKRIPSI
read(psn,n,g);
if psn,g bukan bilangan biner or n < 1 then
   program berhenti dengan pesan kesalahan;
else
   [PXOR,der_g] ← LFSR(g);
   dimensi kode BCH ← n - der_g;
   if banyaknya kolom psn ≠ dimensi kode BCH then
      program berhenti dengan pesan kesalahan;
   else
      katakode ← []; 'inisialisasi'
      cari banyaknya pesan yang akan diencoding;
      jarak_minimum ← n; 'inisialisasi'
      for i ← 1 to banyaknya pesan yang akan diencoding do
         init[1..der_g ] ← [0, 0, ..., 0]; 'inisialisasi LFSR'
         'setiap pesan diproses dari posisi terakhir'
         for j ← banyaknya kolom psn downto 1 do
            pos_trkhr ← init[der_g] XOR psn[i, j];
            init ← geser LFSR secara siklik 1 posisi ke kanan;
            init[1] ← pos_trkhr;
            for k ← 1 to banyaknya posisi XOR - 1 do
```

Lampiran 26. Lanjutan

```
            init[PXOR[k] + 1] ← init[PXOR[k] + 1] XOR pos_trkhr;
          endfor
        endfor
        bbt ← Bobot([init psn(i,:)]);'gabungan matriks init dan psn membentuk
                                    sebuah katakode'
        if bbt ≠ 0 and bbt ≤ jarak_minimum then
          jarak_minimum ← bbt;
        endif

        katakode ← [katakode;init psn(i,:)];
      endfor

      if banyaknya pesan yang dienCoding hanya satu then
        jarak_minimum ← bobot(katakode);
      endif
    endif
    write(katakode);
    write(jarak_minimum);
    write(PXOR);
endif
```

Lampiran 27. Hasil algoritma Encoding_BCH semua kemungkinan katakode untuk panjang kode $n = 15$, dimensi = 5, $g(x) = 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}$

katakode =

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

jm = 7
pxor = 1 2 4 5 8 10