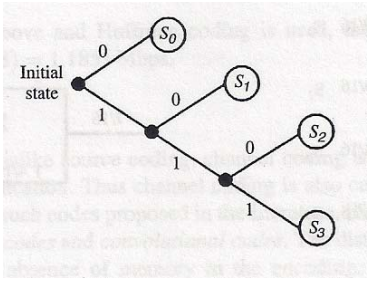
Kode Sumber dan Kode Kanal

Sulistyaningsih, 05912-SIE
Jurusan Teknik Elektro Teknologi Informasi FT UGM,
Yogyakarta

8.2 Kode Awalan

Untuk sebuah kode sumber menjadi praktis digunakan, kode harus dapat dikodekan kembali dengan unik. Kami sangat tertarik pada kelas penting dari kode-kode yang dikenal sebagai *kondisi awal.* Beberapa urutan dibuat dari bagian inisial dari *code word* yang disebut awalan dari *code word*. Untuk contoh, (0),(01),(011) dan (0110) adalah semua awalan dari *code word* (0110). Sebuah awalan didefinisikan sebagai kode dimana tanpa *code word* merupakan awalan dari beberapa *code word* lain.

Tabel 8-1 digambarkan tiga sumber kode untuk pengkodean dari $A^{(2)}$. Kode I bukan kode awalan, sejak *code word* (0) untuk s_0 merupakan awalan dari *code word* (00) untuk s_1.



Gambar 8-2 Pohon keputusan untuk Kode III dari Tabel 8-1

Dengan cara yang sama, kami menunjukkan kode II adalah bukan kode awalan, tetapi kode III adalah kode awalan dengan sifat penting yang ini selalu dapat dikodekan kembali dengan unik.

Tetapi kebalikannya tidak selalu benar. Sebagai contoh, kode II pada Tabel 8-1 tidak memenuhi kondisi awalan, belum dapat dikodekan kembali dengan unik semenjak bit 1 menunjukkan awal dari setiap *code word* dalam kode.

Untuk menunjukkan bahwa kode awalan selalu dapat dikodekan kembali, kami mengambil kode III pada Tabel 8-1 sebagai contoh. Menganggap bit sumber asli adalah 001011010000. Enkoder mengambil setiap dua bit sebagai sebuah kelompok dan memetakan *code word* yang sama, hasilnya dalam urutan 01101111000. Untuk mengkodekan kembali urutan dari *code word* membangkitkan dari kode awalan, dekoder sumber sederhana mulai pada permulaan dari urutan dan mengkodekan kembali satu *code word* pada waktu dasar di pohon keputusan. Gambar 8-2 menggambarkan pohon keputusan yang cocok untuk kode III pada Tabel 8-1. Pohon terdiri sebuah kondisi inisial dan kondisi empat terminal yang cocok untuk simbol sumber s_0,s_1,s_2 dan s_3. Dekoder selalu mulai dari kondisi inisial. Bit penerima bergerak ke atas ke kondisi terminal s_0 jika 0, atau lainnya bergerak ke bawah ke titik keputusan kedua. Dekoder akan mengulang prosesnya sampai kondisi mengkodekan kembali mencapai kondisi akhir. Sekali kondisi mengkodekan kembali mencapai kondisi terminal, dekoder akan mengeluarkan simbol yang cocok dari kondisi

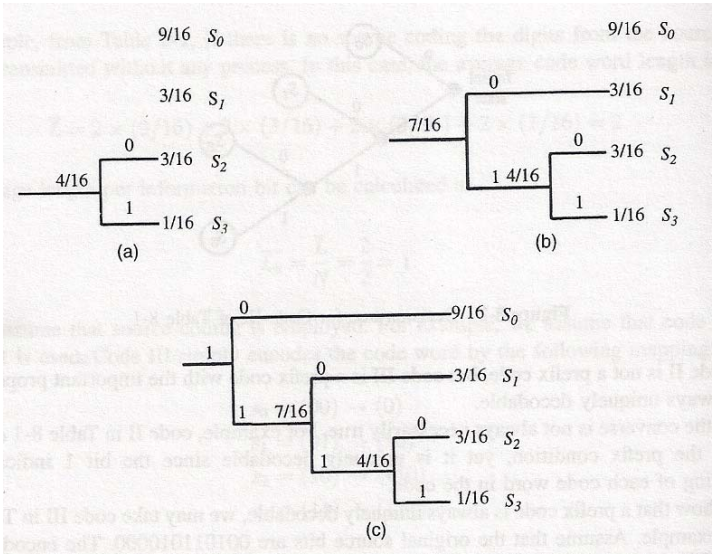
terminal. Kemudian kondisi mengkodekan kembali memasang kembali ke kondisi inisial dan keseluruhan proses mengkodekan kembali ditunjukkan kembali sampai akhir dari rentetan kode.

8.3 Kode Huffman

Kami akan menjelaskan kelas penting yang dikenal sebagai kode Huffman. Ide dasar dari kode Huffman adalah untuk menugaskan simbol dari peluang panjang sebuah *code word* pendek dan menugaskan simbol dari peluang kecil sebuah *code word* panjang. Gagasan kode adalah dapat mengkodekan kembali dengan unik sejak kode awalan. Pokok dari algoritma untuk membuat pohon keputusan tahap demi tahap. Gagasan dari proses kode Huffman sebagai berikut:

1. Simbol dua sumber dari peluang terendah disimbolkan sebagi 0 dan 1.
2. Dua simbol sumber peluang terendah kemudian dikombinasi ke dalam simbol sumber baru yang kemungkinan sama dengan jumlah dari peluang dua simbol asli. Bilangan dari simbol sumber berkurang oleh kombinasi ini.

Prosedur dari step 1 dan step 2 diulang sampai bilangan dari simbol sumber berkurang ke satu.



Gambar 8-3 Susunan dari kode Huffman untuk 4 simbol, (a) tahap pertama, (b) tahap kedua, (c) tahap terakhir

Sebagai contoh, susunan dari empat simbol kode Huffman dengan peluang sumber 9/16,3/16,3/16 dan 1/16 digambarkan pada Gambar 8-3. Tahap pertama ditunjukkan pada 8-3(a), dari simbol dua peluang terendah dengan peluang 3/16 dan 1/16 merupakan kombinasi ke dalam satu simbol baru dengan peluang 3/16 + 1/16 = 4/16. Dua simbol peluang terendah disimbolkan dengan 0 dan 1, berturut-turut. Permasalahannya adalah sekarang mengurangi susunan pohon simbol kode Huffman

dengan peluang sumber 9/16, 3/16 dan 4/16. Dua simbol peluang terendah diantara tiga simbol dengan peluang 3/16 dan 4/16. Gambar 8-3(b) menunjukkan kombinasi dari simbol dengan peluang 3/16 dan 4/16 dan bit disimbolkan, hasilnya dalam simbol baru dengan peluang 7/16.

Sekarang hanya ada dua simbol tertinggal dengan peluang 9/16 dan 7/16. Tahap akhir adalah untuk mengkombinasi dua simbol terakhir dan disimbolkan 0 dan 1, berturut-turut, untuk mereka. Sekarang, kami mempunyai susunan sebuah pohon. *Code word* untuk setiap simbol sumber dapat dibaca dari simbol paling kiri ke kanan mengikuti pohon. Sebagai contoh, untuk simbol s_3, *code word* seharusnya 111 dibaca dari kiri ke kanan. Untuk simbol s_2, *code word* seharusnya 110, dan seterusnya. Susunan kode persisnya kode sama sebagai kode III pada Tabel 8-1. Dengan menggunakan persamaan (8-1), panjang rata-rata *code word* adalah L=1.6875. Panjang rata-rata tiap bit informasi dapat dihitung sebagai Lb=L/2=0.8438. Hasil ini menunjukkan efisiensi dari bit sumber dapat di tingkatkan secara signifikan dengan kode Huffman.

Untuk menunjukkan bagaimana meningkatkan efisiensi kode sumber dari sistem komunikasi digital, kami menganggap bahwa sistem pengiriman dengan rata-rata 1 Mbps; itu, kanal enkoder ditunjukkan pada Tabel 8-1 menerima 1Megabit per detik dari enkoder sumber. Sekarang jika tanpa mempekerjakan enkoder sumber, pengiriman data rata-rata adalah 1 Mbps. Jika sumber mempunyai statistik ditunjukkan di atas dan kode Huffman digunakan, rata-rata pengiriman dapat ditingkatkan menjadi (1/0.8438)=1.1851 Mbps.

8.4 Channel Coding

Seperti yang dijelaskan sebelumnya, tidak seperti *Source coding*, kode saluran menambah redudansi untuk meningkatkan keandalan komunikasi. Sehingga saluran coding disebut juga sebagai *error correcting code*. Terdapat banyak aturan-aturan yang berbeda pada literature. Secara historis, kode ini telah di klasifikasi menjadi *block codes* dan *convulutional codes*. Fitur yang membedakan diantara keduanya adalah terdapat atau tidak terdapatnya memori pada bagian *encoding*. *convulutional codes* memiliki memori pada encoder sedangkan pada *block codes* tidak terdapat.

Block codes menerima informasi dalam blok *k*-bit yang berurutan. Setiap blok, menambah *n-k redundant* bit yang berkaitan dengan aljabar bit pesan *k*. oleh karena itu, ini membuat secara keseluruhan blok encoded dari bit *n*, dengan $n > k$. *Blok* bit n disebut juga sebagai *code word*, dan *n* disebut sebagai panjang blok dari kode. Sederhananya, kita menyebut seperti *block code (n,k) code block*.

Kode koreksi kesalahan secara khusus sangat bermanfaat ketika saluran memiliki level noise yang tinggi. Seperti noise yang disebabkan oleh transmisi yang tidak dapat diandalkan sama sekali. Kita mengataka bahwa bit transmisi dalam keadaan error jika pemancar mengirimkan 0 sementara disisi penerimanya menerima 1. Sekarang, jika peluang dari kesalahan bit adalah besar, transmisi dari informasi tidaklah mudah tanpa adanya kode *channel coding*.

Contoh sederhana dari kode saluran adalah pengulangan kode. Setiap kode word dari pengulangan kode mewakili 1 bit informasi, yaitu $k = 1$. Enkoder tersebut hanya mengulangi bit informasi $n - k = n - 1$ kali. Contoh, misalkan $n = 3$. Jika kita ingin mengirimkan 1, selanjutnya kita dapat menambahkan dua tambahan 1 sebagai bit *redundant*, hasil kode wordnya (1,1,1); jika kita ingin mengirimkan 0, selanjutnya kita dapat menambahkan dua tambahan 0 sebagai bit *redundant*, hasil kode wordnya (0,0,0). Pengulangan kode panjang ke *n* dapat benar selama kesalahannya memenuhi $t = (n-1)/2$. Pada kasus ini, $n = 3$, sehingga kita dapat memperbaiki satu kesalahan dalam tiga bit kode. Contoh, asumsikan bahwa (111) ditransmisikan dan kata yang diterima adalah (011). Terdapat satu bit yang salah pada posisi awal. Sekarang, pada sisi penerima, decoder akan menemukan kode word, antara (000) dan (111), yang terdekat untuk menerima word (011). Kode word memiliki dua posisi yang berbeda dari word yang diterima (011) dan kode word (111) hanya memiliki satu posisi yang berbeda dari (011). Oleh karena itu, pada kasus ini, decoder akan memutuskan bahwa kode word yang akan ditransmisikan adalah (111), dan dapat disimpulkan bahwa decode bit informasi adalah 1. Hal ini mudah untuk diverifikasi bahwa docoder dapat satu bit yang salah dengan tiga bit kode.

8.5. Kemampuan Memperbaiki Kesalahan dan Jarak *Hamming*

Kita telah mendemonstrasikan ide yang sederhana dari kode perbaikan kesalahan dengan menggunakan kode pengulangan. Sekarang permasalahannya adalah bagaimana mengukur kemampuan memperbaiki kesalahan dari kode blok. Jawaban untuk pertanyaan tadi adalah jarak *hamming* dari kode blok. Misalkan $c_1 = (c_{1,1}, c_{1,2}, \ldots, c_{1,n})$ dan $c_2 = (c_{2,1}, c_{2,2}, \ldots, c_{2,n})$ menjadi dua n-tuple vektor biner. Jarak Hamming antara c_1 dan c_2 dinotasikan sebagai $H(c_1, c_2)$, adalah didefinisikan sebagai jumlah digit dari c_1 dan c_2 yang berbeda. Contoh, jarak hamming antara (010) dan (100) adalah 2, karena posisi pertama dan kedua adalah berbeda.

Menetapkan C sebagai kumpulan kode word. Contoh, untuk kode pengulangan dengan panjang 3, $C = \{(000),(111)\}$. Catatan bahwa kode mungkin terdiri lebih dari dua kode word. Jarak minimal dar *C*, dinotasikan sebagai d_{min}, adalah didefinisikan sebagai jarak minimal jarak Hamming antara kode word yang berbeda dari C, yaitu :

$$d_{min} = \min_{\substack{c_1 \epsilon C, c_2 \epsilon C \\ c_1 \neq c_2}} H(c_1, c_2). \qquad (8\text{-}3)$$

Contoh 8-1

Berdasarkan kode berikut (disebut A) :

$$0 \rightarrow 000$$
$$1 \rightarrow 111.$$

Nilai d_{min} untuk kode ini adalah 3. Sekarang berdasarkan kode yang lainnya (B), dapat dilihat sebagai berikut :

$$00 \rightarrow 000$$
$$01 \rightarrow 011$$
$$10 \rightarrow 101$$
$$11 \rightarrow 110.$$

Dapat dilihat bahwa d_{min} untuk kode ini adalah 2.

Pada sisi penerima, setelah menerima word *x*, decoder akan mendapat kode word *D*(*x*) antara *C* yang terdekat pada jarak Hamming sampai *x*. Secara eksplisit, decoder akan mendapat kode word

$$c = D(x) = \arg\min_{c' \in C} H(x, c'). \qquad (8\text{-}4)$$

Hal ini disebut sebagai jarak minimal dekoder.

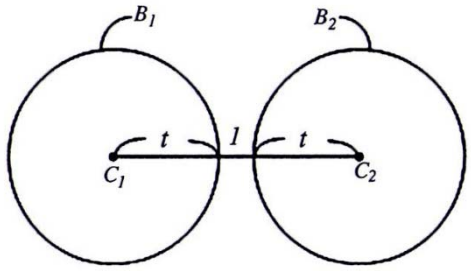
Contoh 8-2

Berdasarkan kode A pada contoh di atas. Misalkan penerima mendapat 010, ini akan dikodekan sebagai 0. Jika ini menerima 110, ini akan dikodekan sebagai 1. Misalkan sekarang bahwa kode B adalah digunakan dan penerima mendapat 001. Decoder tidak dapat secara khusus mendekode pesan. Dapat dikatakan bahwa hanya pesan asli yang baik adalah 00,01 atau 10.

Untuk blok kode (*n*,*k*), terdapat kode word 2^k untuk panjang ke *n*, yang dilambangkan sebagai c_1, c_2, $c_2{}^k$ dalam *C*. Menentukan $B_i = \{x : D(x) = c_i\}$. Sudah jelas bahwa, berdasarkan jarak minimal pengkodean, Kemungkinan semua kombinasi *n*-tuple 2^n dapat dibagi menjadi 2 bagian, $B_1, B_2, \ldots B_2{}^k$, sehingga $B_i \cap B_j = \emptyset$, untuk $i \neq j$. Dekoder akan menampilkan kode word c_i jika dan hanya jika $x \in B_i$.

Secara intuitif, jarak minimal kode yang besar, member kemampuan koreksi kesalahan kode menjadi lebih baik. Contoh 8-2 secara tidak langsung telah menunjukkan hal ini. Kemampuan koreksi kesalahan dari kode blok didefinisikan sebagai jumlah maksimal kesalahan yang dapat diperbaiki oleh decoder. Untuk kode jarak minimal d_{min}, kemampuan koreksi kesalahan adalah

$$t = (d_{min} - 1)/2. \qquad (8\text{-}5)$$



Gambar 8-4 Jarak Hamming minimal dan kemampuan koreksi kesalahan.

Hal ini dapat dijelaskan pada gambar 8-4, yang memiliki dua jarak minimal kode words. c_1 dan c_2 dari C. Di sekeliling lingkaran kode word mewakili masing-masing wilayah B_1 dan B_2. Jika word *x* yang diterima berada dalam B_1, yang berarti bahwa c_1 adalah kode word terdekat ke *x*. Pada kasus ini, decoder akan menampilkan kode word c_1 yang telah dikodekan. Jika word *x* yang diterima mendekati B_2, selanjutnya decoder akan menampilkan kode word c_2 yang telah dikodekan. Karena jarak Hamming antara c_1 dan c_2 adalah d_{min}, jumlah maksimal kesalahan yang dapat dikoreksi pada B_1 atau B_2 menjadi $(d_{min} - 1)/2$; sebaliknya B_1 dan B_2 akan *overlap*. Misalnya, untuk kode A pada contoh 8-1, yang merupakan kode panjang 3 kali pengulangan, kemampuan koreksi kesalahan dari kode ini adalah 1, karena pada kasus ini d_{min}= 3.

Pengkodean Sumber dan Pengkodean Kanal

Anggun Fitrian Isnawati, 06244-SIE
Jurusan Teknik Elektro Teknologi Informasi FT UGM,
Yogyakarta

8.6 KODE HAMMING

Kita akan menggunakan kode yang sudah sangat terkenal yang disebut kode Hamming, sebagai sebuah contoh. Panjang n dan jumlah bit informasi k dari suatu kode Hamming mempunyai bentuk $n = 2^m - 1$ dan $k = 2^m - 1 - m$ untuk integer positif $m \geq 3$. Kode Hamming mempunyai kemampuan mengkoreksi satu bit *error* dalam n bit yang terkodekan.

Sebagai contoh, misal m = 3, maka kita mempunyai sebuah kode Hamming (7,4). Terdapat 16 kata kode (*codeword*) dalam kode Hamming (7,4). Anggap (i_1, i_2, i_3, i_4) sebagai bit-bit informasi. *Codeword* dari kode Hamming mempunyai bentuk (i_1, i_2, i_3, i_4, p_1, p_2, p_3), dimana p_1, p_2, dan p_3 adalah bit-bit tambahan yang ditambahkan oleh *encoder*. Bit-bit tambahan tersebut juga disebut sebagai bit cek paritas (*parity check bit*). Nilai dari p_1, p_2, dan p_3 ditentukan berdasarkan aturan pengkodean dari kode Hamming (7,4). Nilai p_1, p_2, dan p_3 harus ditambahkan sedemikian hingga vektor berikut berisi sejumlah genap bit 1:

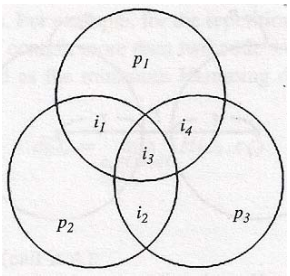
$$(i_1, i_3, i_4, p_1) \qquad (8\text{-}6)$$

$$(i_1, i_2, i_3, p_2) \qquad (8\text{-}7)$$

$$(i_2, i_3, i_4, p_3) \qquad (8\text{-}8)$$

Sebagai contoh, jika (i_1, i_2, i_3, i_4) = (1,0,0,1), maka kita mempunyai nilai $p_1 = 0$ untuk meyakinkan bahwa vektor (i_1, i_3, i_4, p_1) terdiri atas sejumlah genap bit '1'. Kita dapat menyebut p_1 sebagai bit cek paritas untuk i_1, i_3 dan i_4. Dengan cara yang sama, kita mendapatkan $p_2 = 1$ dan $p_3 = 1$. Dengan demikian, *codeword* yang terkodekan menjadi (1,0,0,1,0,1,1).

Kita akan menggunakan gambar 8-5 untuk mengilustrasikan proses pengkodean. Terdapat tiga lingkaran yang merepresentasikan vektor dari persamaan (8-6), (8-7) dan (8-8). Sebagai contoh, lingkaran yang paling atas mencakup empat komponen (i_1, i_3, i_4, p_1). Nilai dari p_1, p_2, dan p_3 harus ditambahkan sehingga setiap lingkaran berisi sejumlah bit 1 genap.



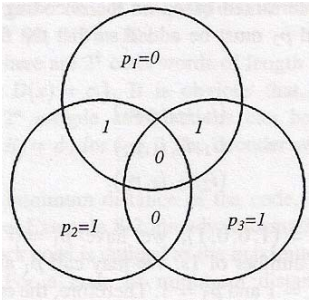
Gambar 8-5 Diagram untuk *encoder* kode Hamming

Sebagai contoh lain, jika (i_1, i_2, i_3, i_4) = (0 1,0,0,1) maka pertama-tama kita harus menempatkan bit-bit informasi ke dalam posisi yang tepat sebagaimana ilustrasi

pada gambar 8-5, hasilnya terlihat pada diagram dalam gambar 8-6. Sekarang, kita akan menentukan nilai dari p_1, p_2, dan p_3 berdasarkan aturan bahwa setiap lingkaran harus berisi sejumlah bit 1 genap. Hasilnya lagi-lagi $p_1 = 0$, $p_2 = 1$ dan $p_3 = 1$.

Seluruh *codeword* dari kode Hamming (7,4) yang mungkin, diberikan pada tabel 8-2 berdasarkan aturan pengkodean sebagaimana ilustrasi di atas. Hal ini membuktikan bahwa jarak minimum dari kode ini adalah 3 dengan cara membandingkan semua pasangan *codeword*. Dengan demikian, kode Hamming (7,4) dapat mengkoreksi satu *error*.

Sekarang kita akan mendemonstrasikan sebuah skema *decoding* dan menunjukkan bagaimana mengkoreksi satu *error* atas penerimaan *codeword*. Asumsi bahwa *codeword* yang ditransmisikan adalah c = (1,0,0,1,0,1,1) dan terjadi *error* pada posisi kedua, maka hasil dari *codeword* yang diterima adalah x = (1,1,0,1,0,1,1). Permasalahan dalam proses *decoding* adalah bagaimana menemukan posisi *error* dengan menggunakan aturan pengecekan paritas sederhana (*simple parity check rule*), sebagaimana yang dinyatakan dalam vektor pada persamaan (8-6), (8-7) dan (8-8). Gambar 8-5 digunakan untuk membantu mengidentifikasi aturan pengecekan paritas.



Gambar 8-6 Pengkodean (i_1, i_2, i_3, i_4) = (1,0,0,1)

Tabel 8-2 Bit-bit informasi dan *codeword* dari kode Hamming (7,4)

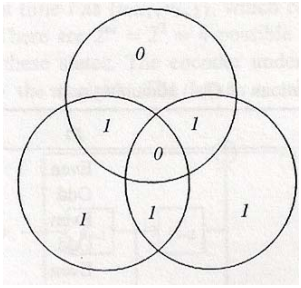
Information Bits	Codeword	Information Bits	Codeword
(0,0,0,0)	(0,0,0,0,0,0,0)	(1,0,0,0)	(1,0,0,0,1,1,0)
(0,0,0,1)	(0,0,0,1,1,0,1)	(1,0,0,1)	(1,0,0,1,0,1,1)
(0,0,1,0)	(0,0,1,0,1,1,1)	(1,0,1,0)	(1,0,1,0,0,0,1)
(0,0,1,1)	(0,0,1,1,0,1,0)	(1,0,1,1)	(1,0,1,1,1,0,0)
(0,1,0,0)	(0,1,0,0,0,1,1)	(1,1,0,0)	(1,1,0,0,1,0,1)
(0,1,0,1)	(0,1,0,1,1,1,0)	(1,1,0,1)	(1,1,0,1,0,0,0)
(0,1,1,0)	(0,1,1,0,1,0,0)	(1,1,1,0)	(1,1,1,0,0,1,0)
(0,1,1,1)	(0,1,1,1,0,0,1)	(1,1,1,1)	(1,1,1,1,1,1,1)

Pertama-tama kita harus menempatkan *codeword* yang diterima x = (1,1,0,1,0,1,1) menurut posisi yang tepat sebagaimana ditunjukkan pada gambar 8-5, hasilnya terlihat pada diagram dalam gambar 8-7. Jika tidak terdapat *error*, semua lingkaran seharusnya berisi sejumlah bit 1 genap. Akan tetapi jika ada satu bit yang diterima mengalami *error*, maka jumlah bit 1 tidak akan berjumlah genap pada beberapa lingkaran. Sekarang kita menghitung jumlah bit 1 di setiap lingkaran seperti pada gambar 8-7. Lingkaran teratas yang sesuai dengan vektor (8-6) berisi sejumlah bit 1 genap. Kita dapat menyimpulkan pada tingkat ini bahwa bit-bit di antara (i_1, i_3, i_4, p_1) sudah benar. Lingkaran sebelah kiri yang sesuai dengan vektor (8-7) berisi sejumlah bit 1 ganjil. Kita dapat menyimpulkan pada tingkat ini bahwa bit-bit di antara (i_1, i_2, i_3,

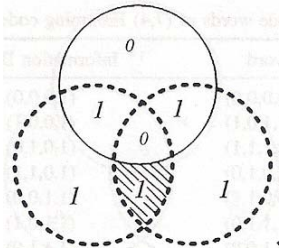
p_2) terdapat *error*. Lingkaran sebelah kanan yang sesuai dengan vektor (8-8) juga berisi sejumlah bit 1 ganjil. Kita dapat menyimpulkan pada tingkat ini bahwa bit-bit di antara (i_2, i_3, i_4, p_3) terdapat *error*.

Dapat disimpulkan bahwa:

- Tidak terdapat *error* pada (i_1, i_3, i_4, p_1)
- Terdapat satu *error* pada (i_1, i_2, i_3, p_2)
- Terdapat satu *error* pada (i_2, i_3, i_4, p_3)



Gambar 8-7 *Decoding* terhadap penerimaan kata (1,0,0,1,0,1,1)



Gambar 8-8 Identifikasi lokasi *error*

Ingat bahwa kita telah mengasumsikan hanya ada satu bit *error* pada *codeword* yang diterima. Permasalahannya sekarang adalah bagaimana mengidentifikasi lokasi *error* berdasarkan induksi di atas. Pada gambar 8-8, lingkaran yang terdapat *error* digambarkan dengan garis putus-putus. Kita mungkin menemukan bahwa bit-bit yang merupakan interseksi antara dua lingkaran tersebut adalah i_2 dan i_3. Akan tetapi, dari deduksi di atas (point 1), i_3 sudah benar. Akhirnya kita simpulkan bahwa i_2 seharusnya merupakan *error* dan cara mengkorekasi *error* ini adalah dengan mengganti $i_2 = 1$ menjadi $i_2 = 0$.

Tabel 8-3 menunjukkan seluruh paritas yang mungkin dan posisi *error* yang sesuai. *Decoder* dapat dengan mudah menghitung jumlah bit 1 di antara vektor (8-6), (8-7) dan (8-8) terhadap penerimaan kata dan mengkoreksi *error* berdasarkan tabel 8-3.

Persamaan (8-6) sampai (8-8) bukan hanya persamaan untuk kode Hamming (7,4). Anggap persamaan berikut:

(i_2, i_3, i_4, p_1) (8-9)

(i_1, i_3, i_4, p_2) (8-10)

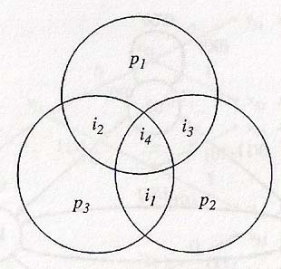
(i_1, i_2, i_4, p_3) (8-11)

Ketiga persamaan ini dapat diilustrasikan dalam gambar 8-9, dengan cara yang sangat mirip pada gambar 8-6.

Secara umum, setiap p_i seharusnya dihubungkan dengan kelipatan 3 yang berisi i_1, i_2, i_3, dan i_4 serta kelipatan 3 yang disesuaikan dengan p_1, p_2, dan p_3 harus jelas.

Tabel 8-3 Paritas dan lokasi *error* dari kode Hamming (7,4)

p_1	p_2	p_3	Error Position
Even	Even	Even	No error
Even	Even	Odd	7
Even	Odd	Even	6
Even	Odd	Odd	2
Odd	Even	Even	5
Odd	Even	Odd	4
Odd	Odd	Even	1
Odd	Odd	Odd	3



Gambar 8-9 Diagram lain dari *encoder* kode Hamming