Gray Code dan Aplikasinya

Unggul Satrio Respationo – NIM : 13506062
Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail: if16062@students.if.itb.ac.id

Abstrak *- Makalah ini membahas Gray-code sebagai suatu teori yang membawa banyak manfaat terhadap perkembangan teknologi komputasi dan teori kombinatorial dan berbagai aplikasinya. Gray-code atau juga dikenal dengan reflected binary code dinamakan setelah Frank Gray, adalah sistem penomoran biner dimana dua nilai yang bersebelahan hanya memiliki tepat satu digit beda. Pada awalnya Gray-code digunakan untuk mencegah keluaran yang palsu dari suatu sinyal elektromekanik. Dewasa ini, Gray-code digunakan secara luas untuk memfasilitasi koreksi galat pada komunikasi digital.*

Kata Kunci: *Gray-code, binary reflected code, Hamming distance, encoding, decoding, binary*

1. PENDAHULUAN

Pada awalnya *Frank Gray* memperkenalkan *reflected binary code* dalam paten aplikasinya tahun 1947. Dia memberikan nama berawal dari fakta bahwa kode ini "mungkin dibentuk dari kode biner yang konvesional dengan urutan proses yang terbalik". Kode ini diberi nama *Gray-code* oleh orang lain yang kemudian menggunakannya.
Sebelumnya *Gray-code* diaplikasikan pada teka-teki matematika sebelum dikenalkan pada *engineer.*

2. GRAY CODE

maju kemudian sebaliknya : 0, 1, 1, 0. Tambahkan 0 pada setengah pertama dan 1 pada setengah kedua menjadi : 00, 01, 11, 10. Teruskan dengan : 00, 01, 11, 10, 10, 11, 01, 00 akan menghasilkan : 000, 001, 011, 010, 110, ... (Sloane's A014550). Setiap iterasi akan menggandakan jumlah dari kode.

Gambar 1. Representasi *Gray-code* dalam pohon biner

Gambar 1 menunjukkan representasi dalam biner 255 dan 511 *Gray-code* pertama. *Gray-code* dalam beberapa bilangan integer non-negatif ada dalam tabel berikut

Untuk mendapatkan *Gray-code* dari bilangan biner $d_1 d_2 \ldots d_n$, kita mulai dari digit paling kanan d_n(digit terakhir atau digit ke-*n)*. Jika d_{n-1} adalah 1, ubah d_n dengan $1 - d_n$, jika tidak biarkan saja. Teruskan untuk d_{n-1} sampai dengan d_1 dengan mengasumsikan d_0 adalah 0. Bilangan yang dihasilkan $g_1 g_2 \ldots g_n$ adalah *reflected binary code* atau *Gray-code.*
Untuk mengubah *Gray-code* $g_1 g_2 \ldots g_n$ menjadi bilangan biner, dimulai lagi dari digit ke-*n*, dan lakukan penghitungan

$$\Sigma_n \equiv \sum_{i=1}^{n-1} g_i (mod\ 2)$$

Jika Σ_n adalah 1, tukar g_n dengan $1 - g_n$, jika tidak biarkan saja. Selanjutnya hitung

$$\Sigma_{n-1} \equiv \sum_{i=1}^{n-2} g_i (mod\ 2)$$

Dan seterusnya sehingga didapatkan $d_1 d_2 \ldots d_n$ yang merupakan bilangan biner yang berkorespondensi dengan *Gray-code* $g_1 g_2 \ldots g_n$ sebelumnya.
Kode ini dikatakan *reflected* karena bisa dibentuk dengan cara yang *reflected*. Ambil *Gray-code* 0, 1. Tuliskan

0	0	5	111	10	1111
1	1	6	101	11	1110
2	11	7	100	12	1010
3	10	8	1100	13	1011
4	110	9	1101	14	1001

Tabel 1. *Gray-code* unutk beberapa integer non-negatif

Berikut akan diberikan algoritma berupa *pseudo-code* untuk menghasilkan *Gray-code* dari bilangan biner (*encode) :*

```
Let B[n:0] be the input array of bits
in the usual binary representation,
[0] being LSB
Let G[n:0] be the output array of
bits in Gray code
   G[n] = B[n]
   for i = n-1 downto 0
     G[i] = B[i+1] XOR B[i]
```

Dan algoritma untuk mengubah *Gray-code* menjadi bilangan biner (*decode)* diberikan sebagai berikut :

```
Let G[n:0] be the input array of bits
in Gray code
 Let B[n:0] be the output array of
bits in the usual binary
representation
   B[n] = G[n]
   for i = n-1 downto 0
     B[i] = B[i+1] XOR G[i]
```

Gray-code ini berhubungan erat dengan solusi pada berbagai permasalahan seperti *Tower of Hanoi*, *Baquenaudier*, seperti juga pada solusi *Hamiltonian Circuits* padagraf *Hypercube.*

3. JENIS KHUSUS GRAY CODE

Dalam prakteknya *Gray-code* hampir selalu merujuk kepada *binary-reflected Gray-code (BRGC)*. Akan tetapi para matematikawan menemukan jenis lain dari *Gray-code.* Seperti BRGC, tiap jenis ini terdiri dari *words*, dimana tiap word berbeda dengan berikutnya hanya sebanyak tepat 1 digit (tiap word mempunya *Hamming distance* 1 terhadap word berikutnya).

3.1 n-ary Gray Code

Salah satunya adalah *n-ary Gray-code* atau juga dikenal sebagai *non-boolean Gray-code*. Seperti pada namanya, kode ini tidak menggunakan boolean dalam *encoding.*
Sebagai contoh 3-*ary (ternary) Gray-code* menggunakan nilai {0, 1, 2}. *(n-k) Gray-code* adalah *n-ary Gray-code* dengan *k*-digit. *(n-k) Gray-code* bisa dibentuk dengan cara rekursif seperti pada BRGC, atau juga dengan cara iteratif. Algoritma *pseudo-code* untuk menghasilkan *(n-k) Gray-ode* secara iteratif diberikan oleh Dah-Jyu Guan sebagai berikut :

```
int n[k+1]; // menyimpan maksimum
untuk tiap digit
int g[k+1]; // menyimpan Gray-code
int u[k+1]; // menyimpan +1 atau -1
untuk tiap elemen
int i, j;
// inisialisasi nilai
for(i = 0; i <= k; i++)
{
        g[i] = 0;
        u[i] = 1;
        n[i] = n;
}
// generate codes
while(g[k] == 0)
{
        i = 0;
        j = g[0] + u[0];
        while((j >= n[i]) || (j < 0))
        {
                u[i] = -u[i];
                i++;
                j = g[i] + u[i];
        }
        g[i] = j;
}
// g[i] now holds the (n,k)-Gray code
```

3.2 Beckett-Gray Code

Jenis lain yang menarik dari *Gray-code* adalah *Beckett-Gray code*. Beckett-Gray code berasal dari nama seorang penulis sandiwara Irlandia bernama *Samuel Beckett* yang secara khusus tertarik kepada sifat simetri. Salah satu sandiwaranya adalah "*Quad"* dibagi menjadi 16 periode waktu.
Dalam sandiwaranya ini, di tiap akhir periode, Beckett menginginkan agar 1 dari 4 aktor masuk atau keluar panggung. Beliau menginginkan sandiwara dimulai dengan panggung yang kosong dan diakhiri dengan pangggung yang kosong pula. Dan beliau juga menginginkan agar setiap sub-himpunan aktor muncul di panggung tepat 1 kali. Jelas sekali, bahwa aktor tersebut dapat direpresentasikan sebagai 4-bit *binary Gray-code*. Tetapi Beckett juga menambahkan batasan dalam *scripting* yaitu agar aktor pertama yang keluar tiap periode adalah aktor yang telah paling lama berada di atas panggung. Maka aktor tersebut dapat juga direpresentasikan sebagai *first in, first out (*struktur data *queue)*. Akan tetapi Beckett tidak dapat menemukan Beckett-Gray code yang dapat emenuhi keinginannya itu, melainkan setelah usaha yang melelahkan dengan menuliskan semua urutan yang mungkin menghasilkan tidak ada kode yang mungkin untuk $n = 4$.
Ilmuwan komputer yang tertarik pada matematika dibalik Beckett-Gray code menemukan bahwa untuk mencarinya sangat sulit. Dewasa ini ditemukan kode ini mungkin untuk $n = \{2, 5, 6, 7\}$ dan tidakmungkin untuk $n = \{3, 4\}$.

3.3 Snake-in-the-box Codes

Snake-in-the-box Codes atau *snakes*, adalah sebuah deret dari simpul dari *induced path* yang ada di dalam *n*-dimensi graf *hypercube*, dan *coil-in-the-box codes* atau *coils*, adalah sebuah deret dari simpul dari *induced cycle* di dalam *hypercube.*
Dengan ditampilkan sebagai *Gray-code*, deret tersebut mempunyai sifat untuk mampu mendeteksi setiap kesalahan bit-tunggal. Kode dalam jenis ini pertama kali dielaskan oleh *W. H. Kautz* di akhir tahun 1950-an. Sejak saat itu telah banya penelitian yang dilakukakn unutk mencari kode dengan kemungkinan terbesar dari *codewords n*-dimensi *hypercube.*

4. LOOPLESS GRAY CODE

Suatu algoritma dikatakan *loopless* jika setelah sebuah objek pertama dihasilkan, objek berikutnya akan didapatkan dalam waktu *O(1)* dalam kasus terburuknya, atau dengan kata lain waktu untuk menghasilkan objek berikutnya adalah konstan tidak bergantung pada ukuran objek *(Ehrlich [1973]).*
Pada bagian ini akan menerangkan contoh aplikasi *loopless Gray-code* dalam berbagai bidang.

4.1 Loopless Gray Code untuk Representasi Minimal Signed-Binary

Sebuah *string* $\ldots a_2 a_1 a_0$ dikatakan *signed-binary representation (SBR)* dari bilangan bulat *n* jika $n = \sum_{k \geq 0} a_k 2^k$ dan $a_k = \{-1, 0, 1\}$ untuk semua nilai *k* bilangan bulat. Sebuah SBR yang minimal mempunyai jumlah bilangan tidak-nol yang paling sedikit. Sebagai contoh, 45 mempunyai 5 SBR minimal : 101101, 110-101, 10-10-101, 10-100-1-1, dan 1100-1-1. Hasil utama kita adalah sebuah algoritma *loopless* yang menghasilkan semua SBR minimal untuk bilangan bulat *n* dalam urutan *Gray-code*. Lihat gambar 2 sebagai contoh. Algoritma ini membutuhkan waktu linier untuk menghasilkan *string* pertama. Setelah itu, hanya *O(1)* waktu yang diperlukan dalam kasus terburuk untuk mengidentifikasi porsi dari *string* yang ada unutuk menghasilkan *string* yang berikutnya.

```
110100-1 -10-1
10-10100-1 -10-1
10-1-100-1 -10-1
10-1-10-1010-1
10-1010-1010-1
11010-1010-1
110011010-1
10-10011010-1
10-100110011
1100110011
11010-10011
10-1010-10011
10-1-10-10011
```

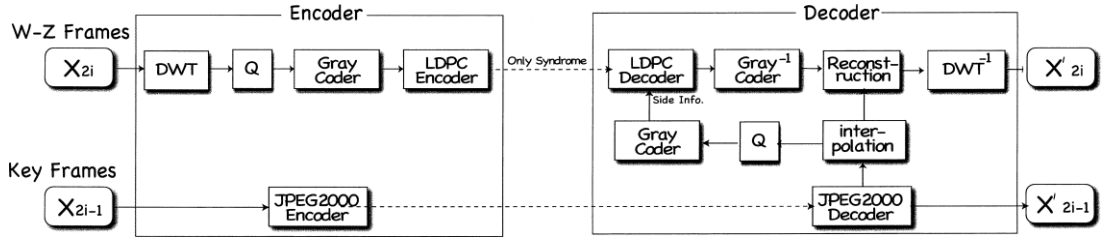
Gambar 2. Sebuah *listing Gray-code* dari SBR minimal untuk 819. *String* berurutan mempunyai beda pada 3 posisi berurut

BRGC-restrict adalah salah satu algoritma yang mampu menyajikan hasil seperti yang diinginkan. Dimana algoritma tersebut disajikan sebagai berikut:

```
BRGC-RESTRICT

INITIALIZE

WHILE true DO
     last <- map(1)
     i <- map(plast)
     IF (i = m + 1) THEN exit

     next(i)

     IF (is_terminal(i)) THEN
          di <- -di
          j <- map(i + 1)
          pi <- pj
          pj <- j

     IF (i /= last) then plast <-
last

Procedure INITIALIZE:

FOR i <- m + 1 DOWNTO 1 DO pi <- i

am <- dm <- 1
IF (EVEN(tm)) THEN rev <- true
ELSE rev ← false

FOR i <- m - 1 downto 1 do

     IF rev = false THEN
          ai <- di <- 1
          IF (EVEN(ti)) THEN rev ←
true
     ELSE
          ai <- ti
          di <- -1
          i <- i - 1
          ai <- di <- 1
          IF (EVEN(ti)) THEN rev <-
false
```

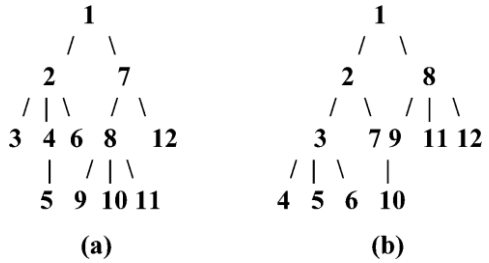
4.2 Loopless Gray Code untuk Pohon Berakar (Rooted Trees)

Beyer dan Hedetniemi [1980] memberikan kostanta waktu-rata pertama dalam algoritma untuk menghasilkan semua rooted treed dengan *n*-simpul. Bagian ini akan menjelaskan kombinatorial *Gray-code* pertama untuk pohon tersebut dan *loopless algorithm* untuk pembangkitannya.
Sebuah pohon berakar (rooted trees) *T* dengan $n \geq 1$ simpul mempunyai sebuah simpul sebagai *akar*-nya dan *k*-subpohon berakar $T_1, T_2, \ldots, T_k$, dengan $n_1 \geq 1, n_2 \geq 1, \ldots, n_k \geq 1$ simpul secara berurutan,dimana $n_1 + n_2 + \cdots + n_k = n - 1$. *Level* dari akar adalah 1, dan level dari simpul yang lain adalah 1 lebih besar daripada *parent*-nya.



Gambar 4. Skema DVC menggunakan sindrom

Sebuah deret level $L(T) = l_1 l_2 \dots l_n$ untuk sebuah pohon berakar T dengan n-simpul didapatkan dengan mentraversal T secara preorder dan l_i adalah level dari simpul ke-i. Karena subpohon T tidak tersusun, banyak deret level yang bergantung pada pohon yang sama. Dari semua deret level yang berhubungan dengan T, representasi kanonik adalah yang terbesar secara *lexicograph*. Pohon berakar dengan 12 simpul ditunjukkan pada gambar 3(a), sebagai contoh, dengan traversal secara preorder, mengindikasikan dengan label bilangan bulat mempunyai deret level sebagai berikut, 1 2 3 3 4 3 2 3 4 4 4 3, ketika representasu kanonik, 1 2 3 4 4 4 3 2 3 4 3 3, berkorespondensi pada gambar 3(b).



Gambar 3. Representasi pohon berakar

Sekarang akan didefinisikan *Gray-code Listing (GCL)* untuk representasi kanonik pada pohon berakar dengan n-simpul deberikan sebagai berikut:

```
Start with l1l2 … ln = 1 2 3 ··· n
and all dir1dir2 … dirn set to down.

if ((n ≤ 2) ← l3 to 2.
while (l3 ≠ 2) next(n).

And next(n) is:
if ((dirn is down and ln > 2) or (dirn
is up and ln < max(l1l2 … ln)))
     if (dirn is down)
          ln ← ln − 1
     else
          ln ← ln + 1
else
     next(n − 1)
     if (l1l2 … ln differs from its
predecessorin exactly one position)
          ln ← max(l1l2 … ln)
     Else
          ln ← 2
          dirn ← up
```

Setiap l_i akan mempunyai arah, dir_i, yaitu atas atau bawah. Fungsi *next(n)* mengubah $l_1 l_2 \dots l_n$ menjadi *successor*-nya didalam GCL. Seperti yang bisa kita lihat, setiap 1 beda dari successor-nya *S(l)* paling banyak pada 3 lokasi. Ini mungkin untuk menemukan kombinatorial *Gray-code* untuk representasi yang mempunyai beda hanya pada 1 atau 2 tempat saja yang masih menjadi pertanyaan.

5. DISTRIBUTING VIDEO CODING MENGGUNAKAN GRAY CODE

Distributed Video Coding (DVC), berdasarkan teori diberikan oleh *Slepian-Wolf* dan *Wyner-Ziv*, menarik perhatian sebagai paradigma baru untuk kompresi video. Beberapa sistem DVC menggunakan kompresi *intraframe* berdasarkan *discrete cosine transform (DCT)*. Sayangnya, sistem DVC yang konvensional mempunyai afinitas yang rendah dengan DCT. Pada bagian ini akan dipaparkan sebuah skema DVC berbasis-*wavelet* yang menggunakan JPEG 2000. Hasilnya, skema ini mempunyai skalabilitas dengan resolusi dan kualitas yang memadai. Skema ini akan menggunakan metode *Gray-code.*

Gambar 4. Skema DVC menggunakan sindrom

Pada gambar 4 ditunjukkan diagram dari skema yang dipakai. Tiap nilai yang terukur dikonversi menggunakan *Gray-coder.* Dengan tujuan untuk mengurangi "galat" pada *correlation channel* antara deret sumber dan informasi sampingan. Keuntungan dari *Gray-code* adalah kita bisa dengan mudah mengkonversinya kembali karena sifat dari *Gray-code* itu sendiri, yaitu *binary-reflected code* yaitu kode yang bisa di *decode* lagi secara terbalik prosesnya seperti saat *encode.*

6. KESIMPULAN

Seperti yang telah banyak disebutkan pada bagian sebelumnya *Gray Code* mempunyai berbagai aplikasi dalam bidang matematika, komputasi, dan juga elektronika. Penggunaanya sangat beragam, dan *Gray-code* ini sendiri masih akan terus dikembangkan sampai sekarang naik oleh matematikawan ataupun imuwan komputer.

Gray-code yang merupakan nama lain yang biasa dianalogikan sebagai *binary-reflected Gray-code* yang sebelumnya hanya diaplikasikan pada teka-teki matematika seperti pada masalah *Hanoi Tower* dan *Hamiltonian cycle* pada *hypercube,* telah berkembang dalam berbagai bidang..

Pada umumnya *Gray-code* digunakan sebagai pendeteksi "galat" pada *encoding* dan *decoding* karena ia hanya memiliki 1 buah digit beda pada tiap 2 bilangan yang berurut sehingga mudah untuk dideteksi. *Gray code* juga bisa digunakan untuk memberi label pada sumbu pada *Karnaugh Maps.* Ketika digunakan pada komputer bahkan *Gray code* bisa digunakan untuk pengalamatan memori program, dimana komputer akan menggunakan sedikit energi karena lebih sedikit *address line* yang berubah saat *program counter* berjalan.

DAFTAR REFERENSI

[1] G.S. Brodal and S. Leonardi (Eds.): ESA 2005, LNCS 3669, pp. 438–447, 2005. Springer-Verlag Berlin Heidelberg 2005

[2] Gray, F. "Pulse Code Communication." United States Patent Number 2632058. March 17, 1953.

[3] Sloane, N. J. A. Sequence A014550 in "The On-Line Encyclopedia of Integer Sequences."

[4] Tonomura, Yoshihide. "Distributed Video Coding Using JPEG 2000 Coding Scheme". NTT Network Inovation Laboratories. November 16, 2006.

[5] Weisstein, Eric W. "Gray Code." From *MathWorld* - A Wolfram Web Resource. http://mathworld.wolfram.com/GrayCode.html. Waktu akses: 26 Desember 2007, pukul: 20.00.