Kode Huffman

Arya Tri Prabawa
Program Studi Teknik Informatika ITB, Bandung 40116, email: if16063@students.if.itb.ac.id

Abstract – *Makalah ini membahas kode Huffman dalam konsep, sejarah singkat, dan aplikasinya. Penggunaan kode Huffman sangat berarti dalam pengelolaan data, baik dalam pengiriman maupun penyimpanannya, sehingga waktu pengiriman dapat dipersingkat dan ruang penyimpanan dapat diperkecil.*

Kode Huffman memanfaatkan kekerapan kemunculan simbol yang akan dimampatkan sehingga simbol-simbol yang seirng muncul dapat direpresentasikan dalam kode-kode yang lebih pendek. Untuk menghindari keambiguan dalam proses decoding*, digunakan kode awalan, di mana kode yang satu bukan merupakan awalan dari kode yang lain.*

Istilah kode Huffman sendiri diambil dari nama penemunya: David A. Huffman. Pada tahun 1951, Huffman dan teman-temannya di kelas Teori Informasi MIT diberi pilihan untuk membuat term paper *mengenai kode biner yang paling efisien atau mengikuti ujian akhir. Hampir menyerah, Huffman akhirnya mendapatkan ide mnggunakan pohon biner yang diurutkan berdasarkan kekerapan. Ia pun berhasil merealisasikan idenya dan bahkan melampaui ide profesornya yang mirip, di mana tidak seperti kode profesornya yang kurang optimal Huffman membangun pohon binernya dari bawah ke atas.*

Kata Kunci*: kode Huffman, Huffman coding, kode variable-length, pohon Huffman, properti prefiks*

1. PENDAHULUAN

Ada beragam jenis kode yang telah digunakan dalam menyatakan huruf atau pesan, yang paling sering digunakan adalah kode Morse, ASCII, dan UNICODE.

Pada tahun 1844, Samuel B. Morse mengirimkan pesan berikut melalui telegraf ciptaannya di antara Washington, DC dan Baltimore, Maryland: “WHAT HATH GOD WROUGHT”. Pesan ini terdiri dari 21 karakter termasuk spasi. Menggunakan karakter

ASCII dalam bentuk integer biner untuk menulis pesan ini membutuhkan 8 bit untuk masing-masing karakternya, sehingga total memerlukan 168 bit. Perbandingan kode Morse dan ASCII dalam merepresentasikan karakter-karakter yang terdapat dalam pesan di atas bisa dilihat pada Tabel 1 di bawah ini:

Tabel 1. Kode Morse dan ASCII

Karakter	Morse	ASCII
W	.--	0101 0111
H	….	0100 1000
A	.-	0100 0001
T	-	0101 0100
G	--.	0100 0111
O	---	0100 1111
D	-..	0100 0100
R	.-.	0101 0010
U	..-	0101 0101
Spasi	(tidak ada)	0010 0000

ASCII adalah contoh dari *fixed-length encoding scheme* di mana panjang setiap karakternya adalah sama, yaitu 8 bit. Sedangkan kode Morse adalah contoh dari *variable-length encoding scheme* dengan panjang representasi karakter yang berbeda-beda. Morse menyadari bahwa kekerapan kemunculan huruf “E” dan “T” dalam teks bahasa Inggris lebih besar daripada huruf “Q” dan “Z” sehingga ia memberikan kode huruf “E” sebuah titik (“.”) dan huruf “T” sebuah garis (“-“), sedangkan huruf “Q” dan “Z” diberikan kode yang lebih panjang, yaitu “--.-“ dan “--..”, begitu pula dengan huruf hidup “A” dan “I”, misalnya, yang diwakili oleh kode yang relatif pendek, yaitu “.-“ dan “..”. Kode Morse ini kemudian dimodifikasi dengan penambahan angka dan tanda baca, seperti juga ASCII yang dikembangkan menjadi UNICODE.

Kita bisa menganggap setiap titik dan garis dalam kode Morse ekuivalen dengan sebuah bit. Karena dalam kode Morse tidak ada representasi karakter “spasi” maka kita asumsikan kode Morse memerlukan sebuah kode dengan panjang 5 bit, sehingga pesan di atas dalam kode Morse hanya memerlukan 62 bit.

Perlu diingat bahwa untuk ke-26 huruf alfabet, panjang kode Morse yang digunakan adalah maksimal 4 bit. Padahal logikanya dalam sebuah representasi dengan panjang yang tetap diperlukan paling tidak 5 bit untuk setiap hurufnya karena panjang 4 bit hanya dapat menghasilkan palaing banyak 16 karakter berbeda. Tidak bisa dipungkiri bahwa dengan menggunakan kode Morse daripada kode ASCII kita dapat menghemat 106 bit atau sekitar 63%. Namun demikian, nyatanya penggunaan kode Morse

menimbulkan masalah dalam proses *decoding*-nya, apa yang disebut *the prefix problem*.

Kelemahan kode *variable-length* adalah sulit untuk dimanipulasi dalam memori komputer, yang dibuat untuk bekerja dengan objek-objek berukuran pasti. Dengan panjang yang tidak pasti, kode Morse juga juga tidak memiliki properti prefiks, dalam arti kode yang satu bukan merupakan awalan pada kode yang lain. Masalah ini dapat ditunjukkan dalam contoh berikut. Bagaimana cara kita men-*decode* kode Morse berikut?

.--.--....---..--

Pesan di atas bisa saja dimulai dengan huruf A (".-"), W (".--"), ataupun P (".--.") karena pada kenyataannya kita dapat melihat bahwa kode huruf A merupakan awalan pada kode huruf W dan P, dan kode huruf W pula adalah awalan pada kode huruf P sehingga sulit untuk memisahkan karakter-karakter yang ada. Bisa saja kita mengasumsikan sebuah karakter pemisah, tetapi hal ini justru akan memperbesar data dan bertentangan dengan tujuan awal kita membuat kode. Lagipula bagaimana bisa kita mengasumsikan sebuah karakter pemisah hanya dengan pilihan titik dan garis?

Pada aplikasi mesin telegraf, seorang operator memberi jarak waktu antara pengiriman karakter yang satu dengna yang lain. Dengan cara ini operator penerima dapat mengetahui di mana karakter yang satu berakhir dan karakter yang lain dimulai. Tidak demikian halnya dengan cara kerja komputer. Tidak ada pemberian jarak waktu pengiriman, yang ada adalah bagian memori yang cukup besar dan hanya menampung nilai 0 dan 1.

Untuk memecahkan masalah ini terciptalah kode Huffman, sebuah kode *variable-length* yang memiliki properti prefiks.

2. SEJARAH SINGKAT

Pada tahun 1951, David A. Huffman dalam kelas Informasi Teori di MIT diberikan pilihan untuk membuat sebuah *term paper* atau mengikuti ujian akhir. Pada saat itu pilihan *term paper* yang diberikan profesor Robert M. Fano adalah tentang menemukan kode biner yang paling efisien. Tidak dapat membuktikan kode apapun yang paling efisien, Huffman hampir menyerah dan mulai belajar untuk mengikuti ujian akhir saja, ketika ia menemukan ide untuk menggunakan pohon biner dengan pengurutan berdasarkan kekerapan dan berhasil membuktikan bahwa cara ini adalah yang paling efisien.

Apa yang dilakukan Huffman melampaui profesornya sendiri, yang bekerja sama dengan pencipta bidang teori informasi Claude Shannon mengembangkan kode yang mirip. Huffman menghindari kesalahan besar dari kode Shannon-Fano yang kurang optimal dengan membangun pohon binernya dari bawah ke atas dan bukan dari atas ke bawah (penjelasan detail mengenai pohon Huffman akan diberikan pada bagian selanjutnya).

Makalah berjudul "A Method for the Construction of Minimum Redundancy Codes" tersebut lalu dipublikasikan oleh Huffman pada tahun 1952 dalam sebuah jurnal profesional untuk *Institute of Radio Engineers*.

3. KONSEP

Mirip dengan kode Morse, kode Huffman juga memanfaatkan kekerapan kemunculan suatu simbol, di mana simbol yang lebih sering muncul diwakili oleh kode yang lebih singkat. Namun, tidak seperti kode Morse, kode Huffman memiliki properti prefiks dalam arti kode yang yang satu bukanlah awalan pada kode yang lain sehingga tidak menimbulkan masalah dalam proses *decoding*-nya. Skema kode Huffman dilakukan dengan membangun sebuah pohon biner dengan jalur berbobot 0 atau 1 di mana elemen-elemen berupa simbol-simbol yang akan dikodekan diurutkan berdasarkan kekerapan kemunculannya.

Untuk mulai membuat kode Huffman untuk pesan "WHAT HATH GOD WROUGHT", terlebih dahulu kita urutkan semua karakter yang ada, termasuk spasi, berdasarkan kekerapan kemunculannya seperti terlihat pada Tabel 2 di bawah ini.

Tabel 2. Urutan karakter berdasarkan kekerapan kemunculan (tabel ke-1)

Kekerapan	Karakter (Subpohon)
1	D
1	R
1	U
2	G
2	O
2	A
2	W
3	T
3	Spasi
4	H

Langkah-langkah selanjutnya dapat dideskripsikan sebagai berikut:

1. Jika daftar hanya terdiri dari satu karakter maka selesai.
2. Jika tidak maka buatlah sebuah subpohon baru dengan menghilangkan dari daftar dan menggabungkan dua subpohon dengan kekerapan terkecil, dan kekerapan subpohon yang baru adalah jumlah dari kekerapan kedua subpohon yang membentuknya.

3. Tambahkan subpohon yang baru ke dalam daftar dengan tetap memperhatikan urutan kekerapannya.
4. Kembali ke langkah no. 1.

Jika langkah-langkah di atas dilakukan terhadap tabel awal maka tabel-tabel selanjutnya yang terbentuk adalah sebagai berikut:

Tabel 3. Urutan karakter berdasarkan kekerapan kemunculan (tabel ke-2)

Kekerapan	Subpohon
1	U
2	(D, R)
2	G
2	O
2	A
2	W
3	T
3	Spasi
4	H

Tabel 4. Urutan karakter berdasarkan kekerapan kemunculan (tabel ke-3)

Kekerapan	Subpohon
2	G
2	O
2	A
2	W
3	(U, (D, R))
3	T
3	Spasi
4	H

Tabel 5. Urutan karakter berdasarkan kekerapan kemunculan (tabel ke-4)

Kekerapan	Subpohon
2	A
2	W
3	(U, (D, R))
3	T
3	Spasi
4	(G, O)
4	H

Tabel 6. Urutan karakter berdasarkan kekerapan kemunculan (tabel ke-5)

Kekerapan	Subpohon
3	(U, (D, R))
3	T
3	Spasi
4	(A, W)
4	(G, O)
4	H

Tabel 7. Urutan karakter berdasarkan kekerapan kemunculan (tabel ke-6)

Kekerapan	Subpohon
3	Spasi
4	(A, W)
4	(G, O)
4	H
6	((U, (D, R)), T)

Tabel 8. Urutan karakter berdasarkan kekerapan kemunculan (tabel ke-7)

Kekerapan	Subpohon
4	(G, O)
4	H
6	((U, (D, R)), T)
7	(Spasi, (A, W))

Tabel 9. Urutan karakter berdasarkan kekerapan kemunculan (tabel ke-8)

Kekerapan	Subpohon
6	((U, (D, R)), T)
7	(Spasi, (A, W))
8	((G, O), H)

Tabel 10. Urutan karakter berdasarkan kekerapan kemunculan (tabel ke-9)

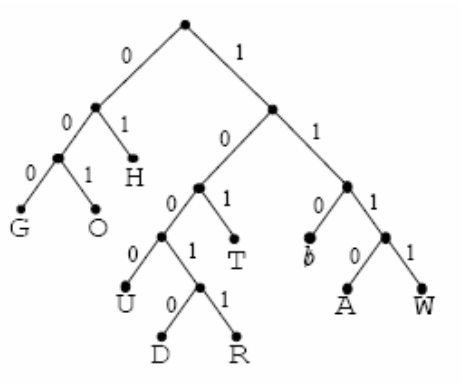
Kekerapan	Subpohon
8	((G, O), H)
13	(((U, (D, R)), T), (Spasi, (A, W)))

Tabel 11. Urutan karakter berdasarkan kekerapan kemunculan (tabel ke-10)

Kekerapan	Subpohon
21	(((G, O), H), (((U, (D, R)), T), (Spasi, (A, W))))

Perlu dilihat bahwa pada subpohon hasil penggabungan, urutan kekerapan kedua subpohon pembentuknya juga diperhatikan. Pada tabel-tabel di atas terlihat bahwa pada sebuah subpohon hasil penggabungan, subpohon pembentuk dengan kekerapan yang lebih kecil ditulis di sebelah kiri dari subpohon pembentuk dengan kekerapan yang lebih besar. Hal ini akan lebih jelas terlihat apabila kita gambarkan dalam bentuk pohon Huffman, yaitu sebuah pohon biner terurut-berdasarkan-kekerapan dengan jalur berbobot 0 atau 1, sebagai berikut

(karakter “spasi” diwakili dengan karakter “ ␣ ”):



Gambar 1: Pohon Huffman

Pohon biner di atas dibangun dengan aturan bahwa jalur ke cabang kiri diberi bobot 0, sedangkan jalur ke cabang kanan diberi bobot 1. Selanjutnya, kode biner setiap karakter diperoleh dengan mengikuti jalur yang dilewati dari akar sampai dengan karakter yang dimaksud. Nilai 0 dan 1 pada jalur yang dilewati membentuk kode biner untuk karakter tersebut. Daftar kode Huffman untuk setiap karakter yang terdapat di dalam pohon Huffman pada Gambar 1 dapat dilihat pada Tabel 12 di bawah ini:

Tabel 12. Kode Huffman

Karakter	Kode Huffman
H	01
Spasi	110
T	101
G	000
O	001
W	1111
A	1110
U	1000
R	10011
D	10010

Dapat dilihat bahwa kode Huffman memiliki properti prefiks, yang berarti tidak ada kode yang merupakan awalan pada kode yang lain. Hal ini juga tergambar dengan jelas apabila kita melihat pohon biner Huffman yang terbentuk, di mana semua karakter terletak pada simpul daun, tidak ada yang terletak pada simpul dalam, sehingga jalur yang dilewati menuju setiap karakter tidak ada yang beririsan. Pesan yang telah dikodekan kini besarnya adalah 68 bit. Memang lebih besar dari hasil pengodean dengan kode Morse, yaitu sebesar 62 bit, tetapi tanpa masalah prefiks yang muncul skema Huffman menjadi lebih efisien.

Ada 2 alternatif dalam menggunakan skema Huffman. Yang pertama adalah dengan mengodekan semua karakter berdasarkan kekerapan kemunculannya sampel teks umum. Alternatif kedua dan yang lebih populer adalah apa yang disebut *adaptive coding*, di mana kekerapan kemunculan yang dimaksud adalah kekerapan kemunculan sebuah karakter dalam *file* yang akan dikodekan saja. Alternatif ini mengharuskan kita membaca *file* dua kali. Pembacaan pertama adalah untuk mengetahui karakter apa saja yang terdapat di dalam *file* beserta kekerapan kemunculan masing-masing karakter yang akan digunakan untuk membangun pohon Huffman. Pada pembacaan kedua kita sudah bisa mengodekan setiap karakter sesuai dengan kode Huffman yang telah didapatkan dari pohon Huffman. Walalupun memerlukan pembacaan yang lebih banyak, altenatif kedua ini menjadi lebih efisien.

Ada beberapa hal yang perlu diperhatikan dalam membuat program untuk menulis dan membaca kode dengan skema Huffman. Pertama, struktur data yang digunakan adalah sebagai berikut:

ATreeNode
- Character (1 byte)
- Frequency (long integer)
- Parent (tree node pointer)
- LeftChild (tree node pointer)
- RightChild (tree node pointer)

Penunjuk Parent dapat digunakan untuk bergerak ke atas dari simpul daun sampai ke akar.

Kedua, jumlah bit yang dipakai tidak selalu keliapatan 8, sehingga byte yang terakhir bisa saja berisi beberapa bit kosong yang dapat membingungkan pembaca kode. Masalah ini dapat diatasi dengan memberikan sebuah *long integer* pada awal *file* yang memberikan informasi berapa banyak karakter yang terdapat di dalam *file* aslinya.

Ketiga, untuk semakin membantu proses pembacaan kode, kita dapat memasukkan pohon Huffman sebagai informasi tambahan pada awal *file* sehingga dalam pembacaan kita hanya perlu membaca pohon Huffman tersebut sampai ke daun dan memperoleh karakter yang diinginkan, terus demikian sampai kita tahu semua karakter telah dibaca berdasarkan *long integer* yang diperoleh pada awal pembacaan.

Sebagai gambaran, berikut ini adalah potongan program dalam bahasa Pascal yang berisi prosedur penulisan kode Huffman dengan simpul akar sebagai parameter input dan fungsi pembacaan kode yang akan mengembalikan pointer ke simpul akar:

```
procedure WriteTheCodeTree(NextNode :
NodePointerType);

begin
{a procedure called WriteABit writes a single bit to
the output stream}
```

```
{a procedure called Write8Bits writes a byte to the
output stream}
if NextNode.LeftChild <> NIL then
  begin
  {If you get here, NextNode is NOT a leaf node}
  {A 1-bit will later (on input) tell us it is not a leaf
  node}
  WriteABit(1);
  WriteTheCodeTree(NextNode.LeftChild);
  WriteTheCodeTree(NextNode.RightChild)
  end
else
  begin
  {If you get here, NextNode is a leaf node}
  {A 0-bit will later (on input) tell us it is a leaf node}
  WriteABit(0);
  Write8Bits(NextNode.Character)
  end
end; {procedure WriteTheCodeTree}



function ReadTheCodeTree : NodePointerType;

var
LeftChild : NodePointerType;
RightChild : NodePointerType;
NewNode : NodePointerType;
Character : byte;

begin
{the function called ReadABit returns the next bit 0 or
1 from the input stream}
{the function Read8Bits returns one ASCII character
from the input stream}
if ReadABit=1 then
  begin
  {the bit just read was a 1 bit}
  LeftChild := ReadTheCodeTree;
  RightChild := ReadTheCodeTree;
  New(NewNode); { allocate memory for a new non
  leaf node}
  NewNode.LeftChild := LeftChild;
  NewNode.RightChild := RightChild;
  ReadTheCodeTree := NewNode
  end
else
  begin
  {the bit just read was a 0 bit}
  Character := Read8Bits;
  New(NewNode); { allocate memory for a new leaf
  node}
  NewNode.Character := Character;
  NewNode.LeftChild := nil;
  NewNode.RightChild := nil;
  ReadTheCodeTree := NewNode
  end
end; {function ReadTheCodeTree}
```

4. KESIMPULAN

Agar proses pengiriman dan penyimpanan data menjadi semakin cepat dan memerlukan lebih sedikit ruang, ukuran data yang besar dapat dimampatkan. Penggunaan kode *fixed-length* seperti ASCII yang menyita banyak tempat dapat diimbangi dengan penggunaan kode *variable-length* yang memerlukan lebih sedikit alokasi memori.

Di mana penggunaan kode Morse sebagai salah satu bentuk kode *variable-length* menimbulkan masalah dalam proses pembacaannya karena tidak memiliki properti prefiks, penggunaan bentuk kode *variable-length* lainnya, yaitu kode Huffman, ternyata berhasil menyediakan skema pengodean yang efisien. Skema pengodean Huffman menggunakan pohon biner terurut-berdasarkan-kekerapan dengan jalur berbobot 0 dan 1, di mana semua karakter yang akan dikodekan terletak pada simpul daun sehingga tidak ada kode yang merupakan awalan pada kode yang lain.

Seiring berjalannya waktu, tentunya semakin banyak pula teknik pemampatan data lain yang dapat digunakan. Namun demikian, skema pengodean Huffman tetap merupakan salah satu yang paling sederhana.

DAFTAR REFERENSI

[1] Kennedy, John, “Huffman Coding”.
[2] http://en.wikipedia.org, Desember 2007.
[3] Munir, Rinaldi, “Diktat Kuliah IF2153 Matematika Diskrit Edisi Keempat”, Informatika ITB, 2006.