Penyandian (*Encoding*) dan Penguraian Sandi (*Decoding*) Menggunakan *Huffman Coding*

Nama : Irwan Kurniawan NIM : 135 06 090
1) Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
Email: if16090@students.if.itb.ac.id

Abstrak

Penyandian (*Encoding*) adalah proses yang penting dalam dunia informatika. Saat ini kemampuan untuk mengirim dan menerima informasi secara cepat sangat dibutuhkan. Semakin besar sebuah data, semakin lama waktu yang diperlukan dalam pengiriman dan semakin besar pula kemungkinan data hilang dalam pengiriman. Oleh karena itu dibutuhkan sebuah cara (proses) untuk mengkompresi data kedalam bentuk sandi – sandi yang lebih optimal tanpa merusak informasi yang dikandung oleh data tersebut.

Huffman Coding adalah salah satu cara penyandian (*Encoding*) Data yang cukup terkenal saat ini. *Huffman Coding* telah digunakan secara luas pada berbagai bahasa pemograman. Hal ini disebabkan keunggulan *Huffman Coding* dalam pengolahan data yang memungkinkan kompresi data mulai dari 22 % hingga 91 % tanpa menyebabkan adanya data yang hilang.

Kata Kunci

Pohon Huffman, Algoritma Huffman, Tabel Kode Huffman, Pengkodean (*Encoding*), Penguraian Kode (*Decoding*).

1. PENDAHULUAN

Huffman Coding dikembangkan oleh seorang mahasiswa MIT, David A. Huffman, ketika ia mengambil gelar Ph.D – nya dengan paper berjudul "A Method for the construction of Minimum – Redundancy Codes".[1]

Komputer menyimpan informasi dalam rangkaian bit 0 – 1 dalam sebuah string biner. Keterbatasan media penyimpanan dan kemampuan transmisi data berukuran besar yang relatif lambat membuat orang berpikir untuk mencari sebuah cara pengkompresian data yang efektif. Efektif berarti cara pengkompresian tersebut tidak boleh menyebabkan adanya data yang hilang dan ukuran hasil kompresi yang relatif lebih kecil dari ukuran semula. Dari beberapa cara pengkodean (*Encoding*) yang ada saat ini, *Huffman Coding* adalah salah satu cara yang banyak dikenal oleh kalangan praktisi informatika karena keefektifan dan kemudahannya.

Keunggulan dari *Huffman Coding* adalah metode pengkodean yang bersifat *universal* sehingga dapat diterapkan pada berbagai jenis data. Metode pengkodean menggunakan *Huffman Coding* juga memberikan hasil yang cukup memuaskan. Percobaan menggunakan *Huffman Coding* Pada 530 *source programs* dengan 4 bahasa pemograman memberikan hasil bahwa kompresi data menggunakan cara pengkodean *Huffman Coding* dapat menghasilkan data dengan ukuran 22 % hingga 91 % dari data semula [3].

Makalah ini akan membahas cara pengkodean menggunakan *Huffman Coding* dan cara penguraian kode dari hasil pengkodean menggunakan *Huffman Coding* tersebut.

2. *HUFFMAN CODING*

2.1. Pohon Huffman

Huffman Coding menggunakan struktur pohon dalam pemrosesannya. Pohon adalah graf tak berarah yang tidak mengandung sirkuit[2]. Di dalam struktur pohon dikenal terminologi *parent* (orang tua) dan *child* (anak). *Parent* (orang tua) yaitu sebuah simpul yang memilki lintasan ke simpul lain dengan tingkatan (*level*) di bawahnya. *Child* (anak) yaitu sebuah simpul yang memiliki lintasan ke simpul lain dengna tingkatan (*level*) di atasnya.

Berdasarkan jumlah anak, pohon dapat dikategorikan sebagai pohon *n – ary*. Pohon dengan orang tua yang hanya memiliki satu anak, disebut pohon uner. Pohon dengan orang tua yang memiliki dua anak, disebut pohon biner, dan seterusnya.

Pohon Huffman menggunakan strukrut pohon biner, yaitu struktur pohon dengan setiap simpul orang tua yang hanya memiliki maksimal 2 simpul anak. Penyusunan data dalam struktur pohon Huffman menggunakan kode awalah (*prefix code*). Kode awalah adalah himpunan kode (biner) dimana anggota kumpulan yang satu bukan merupakan anggota kumpulan yang lain [1].

2.2. Algoritma Huffman

Biasanya sebuah karakter dikodekan dalam kode ASCII (8 bit) atau kode Unicode (16 bit) yang telah memiliki panjang tetap (*fixed - length*). Berbeda dengan cara pengkodean ASCII atau Unicode, pengkodean dengan *Huffman Coding* menggunakan penjang bit yang bervariasi dalam mengkodekan sebuah karakter. Karakter yang memiliki frekuensi kemunculan lebih besar memiliki panjang bit yang lebih pendek, sebaliknya karakter yang memiliki frekuensi kemunculan yang lebih kecil memilki panjang bit yang lebih panjang. Hal ini menyebabkan kode untuk sebuah karakter dengan menggunakan cara pengkodean *Huffman Coding* tidak tetap seperti dalam ASCII *code* atau Unicode.

Seperti dijelaskan sebelumnya pengkodean dengan cara *Huffman Coding* menggunakan kode awalan (*prefix code*) yang direpresentasikan dalam struktur pohon biner. Cara penyusunan ke dalam pohon biner adalah dengan memberikan label '0' untuk cabang kiri dan label '1' untuk cabang kanan. Hal ini dilakukan secara berulang hingga akhirnya terbentuk rangkaian bit yang merupakan kode awalan (*prefix code*).

Langkah – langkah pembentukan pohon Huffman dengan menggunakan Algoritma Huffman :

Algorithm Huffman(X)
// **input**: String X of length n with d distinct characters
// **output**: coding tree for X

```
Compute the frequency function f of
characters in X
Initialize empty priority queue Q of trees
// Fill Q
for each character, c, in X do
Create a single-node binary tree T storing c
Insert T into Q with key f(c)

while Q.size() > 1 do
        f1 = Q.minKey()
        T1 = Q.removeMin()
        f2 = Q.minKey()
        T2 = Q.removeMin()
        Create new binary tree T with left
        subtree T1 and right subtree T2
        Insert T into Q with key f1 + f2

return tree Q. removeMin()
```

2.3. Pengkodean dengan *Huffman Coding*

Berikut ini adalah sebuah contoh cara pengkodean sebuah string. Misalkan kita akan mengkodekan sebuah string "AABCABC".

Dalam ASCII string tersebut akan dikodekan sebagai 010000010100000101000010010000110100000101000001001000011 dengan perincian kode setiap karakter adalah :

- Karakter A dalam kode ASCII 01000001
- Karakter B dalam kode ASCII 01000010
- Karakter C dalam kode ASCII 01000011

Pengkodean string ini ke dalam kode ASCII akan membutuhkan memori sebesar 7 x 8 bit atau sama dengan 7 byte.

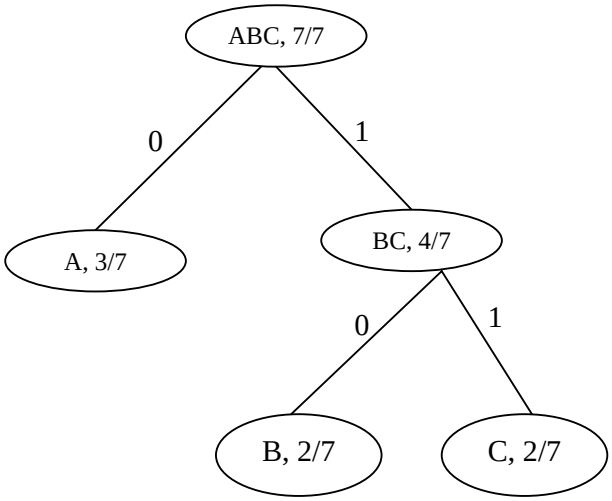
Berikut adalah cara pengkodean string yang sama dengan menggunakan *Huffman Coding*. Berdasarkan algoritma yang telah disebutkan sebelumnya, kita akan menghitung frekuensi kemunculan dari setiap karakter dalam string X.

Tabel yang menunjukkan frekuensi kemunculan setiap karakter tersebut ditunjukkan pada gambar 1.

Simbol	Kekerapan	Peluang
A	3	3/7
B	2	2/7
C	2	2/7

Gambar 1 : Tabel Kekerapan dari kode Huffman untuk string "AABCABC"

Berdasarkan tabel yang ditunjukkan oleh gambar 1 kita dapat menyusun pohon Huffman seperti yang ditunjukkan oleh gambar 2 dengan kode awalan (*prefix code*).



Gambar 2 : Pohon Huffman untuk string "AABCABC"

Berdasarkan pohon Huffman yang ditunjukkan pada gambar 2, kita dapat menentukan Kode Huffman untuk setiap symbol yang ada dalam string "AABCABC". Kode Huffman untuk setiap simbol (karakter) dalam string "AABCABC" ditunjukkan dalam gambar 3.

Simbol	Kode Huffman
A	0
B	10
C	11

Gambar 3 : Kode Huffman untuk karakter dalam string "AABCABC"

Dengan menggunakan Kode Huffman yang ditunjukkan pada gambar 3, maka pengkodean string "AABCABC" dalam kode Huffman adalah : 00101101011 dimana representasi ini hanya membutuhkan memori sebesar 11 bit (1,375 byte).

Bandingkan jumlah memori yang dibutuhkan apabila menggunakan kode ASCII dan kode Huffman. Kode Huffman hanya membutuhkan memori sebesar 19,642% dari kode ASCII !

2.4. Penguraian Kode Huffman (*Decoding*)

Penguraian kode (*Decoding*) adalah sebuah proses untuk menyusun kembali data yang telah dikodekan sebelumnya sehingga informasi yang diterima dapat dibaca dan diolah. Penguraian kode (*Decoding*) ini adalah lawan dari pengkodean (*Encoding*).

Ada 2 cara penguraian kode Huffman. Cara yang pertama adalah menggunakan pohon Huffman sedangkan cara yang kedua adalah menggunakan tabel kode Huffman.

Berikut adalah penjelasan untuk cara pertama, yaitu menggunakan pohon Huffman. Seperti yang dijelaskan sebelumnya, pohon Huffman adalah pohon biner dengan menggunakan kode awalan (*prefix code*). Hal ini memudahkan proses penguraian kode. Langkah – langkah yang dilakukan dalam penguraian kode (*decoding*) menggunakan pohon Huffman adalah sebagai berikut :

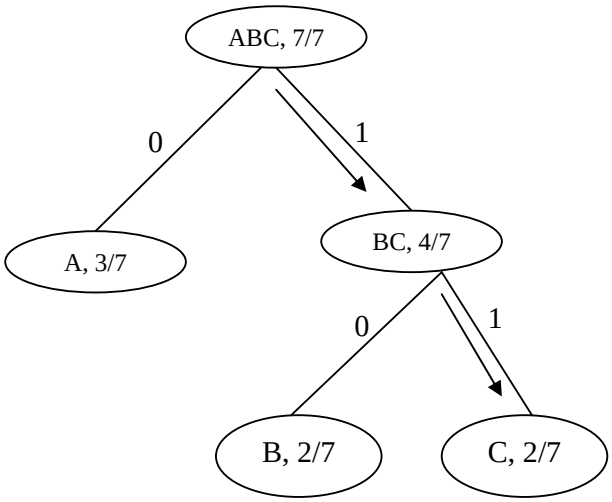
1. Baca bit pertama dari string biner masukan
2. Lakukan traversal pada pohon Huffman mulai dari akar sesuai dengan bit yang dibaca. Jika bit yang dibaca adalah 0 maka baca anak kiri, tetapi jika bit yang dibaca adalah 1 maka baca anak kanan.
3. Jika anak dari pohon bukan daun (simpul tanpa anak) maka baca bit berikutnya dari string biner masukan.
4. Hal ini diulang (traversal) hingga ditemukan daun.
5. Pada daun tersebut simbol ditemukan dan proses penguraian kode selesai.
6. Proses penguraian kode ini dilakukan hingga keseluruhan string biner masukan diproses.

Contoh cara penguraian kode menggunakan pohon Huffman. Dengan menggunakan kode hasil enkripsi yang telah ditunjukkan dalam proses pengkodean (*Encoding*) sebelumnya, akan ditunjukkan cara penguraian kode (*decoding*) menggunakan pohon Huffman.

Hasil pengkodean string "AABCABC" ke dalam string biner adalah 00101101011. Bit pertama dari string biner tersebut adalah 0. dengan menggunakan pohon Huffman yang ditunjukkan dalam gambar 2, ditemukan bahwa anak kiri nya adalah daun yang menyimpan simbol A. dengan melakukan hal yang sama pada bit kedua (angka 0), ditemukan simbol berikutnya adalah A. pada bit ketiga (angka 1), ditemukan bahwa anak kanannya bukanlah sebuah daun. Oleh karena itu harus diambil bit berikutnya

yaitu bit keempat (angka 0). Karena bit keempat adalah 0 maka harus diambil anak kiri. Pada anak kiri tersebut ditemukan sebuah daun yang menyimpan simbol B. Dengan melakukan hal ini secara berulang hingga bit terakhir, maka akan ditemukan bahwa string biner 00101101011 adalah hasil pengkodean (*enkripsi*) dari string "AABCABC".

Untuk memperjelas contoh ini, disertakan dalam gambar 4 cara menentukan sebuah string biner 11 sebagai simbol C berdasarkan pohon Huffman gambar 3.



Gambar 4 : Contoh penentuan string biner 11 sebagai simbol C berdasarkan pohon Huffman gambar 3

Cara kedua untuk menguraikan kode Huffman adalah dengan menggunakan tabel kode Huffman.Oleh karena kode Huffman disusun menggunakan kode awalah (*prefix code*) maka dapat dipastikan bahwa sebuah kode untuk sebuah simbol / karakter yang satu tidak boleh menjadi awalan dari kode / simbol yang lain. Oleh karena itu pastilah string biner yang berisi hasil enkripsi dapat dipisahkan dengan mudah berdasarkan setiap rangkaian bitnya untuk diuraikan menjadi informasi semula. Yang perlu dilakukan hanyalah melihat setiap rangkaian bit yang ditemukan dalam string biner hasil enkripsi di dalam tabel kode Huffman.

Berikut ini akan diberikan contoh cara penguraian kode dengan menggunakan Tabel kode Huffman.

Misalkan akan dilakukan penguraian kode (*decoding*) berdasarkan string biner yang dihasilkan dari proses pengkodean (*encoding*) sebelumnya. String biner yang dihasilkan sebelumnya adalah 00101101011 dengan tabel kode Huffman yang ditunjukkan pada gambar 3.

String biner tersebut dapat dipisahkan menjadi rangkaian bit : 0 0 10 11 0 10 11 . Hal ini dapat dilakukan dengan mudah karena penyusunan rangkaian bit tersebut menggunakan kode awalan (*prefix code*). Setelah string biner tersebut dipisah menjadi rangkaian bitnya, hanya perlu dilihat rangkaian bit yang bersesuaian dengna tabel kode Huffman yang ditunjukkan pada gambar 3. Maka akan dihasilkan string hasil penguraian kode (*dekripsi*), yaitu "AABCABC".

2.5. Notasi O (Big - O) algoritma Huffman

Berdasarkan algoritma Huffman yang ditunjukkan pada upabab 2.2. waktu yang diperlukan untuk menjalankan algoritma ini adalah O(n + d log d) dimana O (d log d) adalah waktu yang diperlukan untuk membangun pohon Huffman dari d jenis karakter dan O (n) adalah waktu yang diperlukan untuk menentukan frekuensi kemunculan setiap karakter dan mengkodekan (*encrypting*) string menggunakan tabel.

3. HASIL DAN PEMBAHASAN

3.1. Perbandingan pengkodean data menggunakan *Huffman Coding* dan kode ASCII

Disini akan dibandingkan hasil pengkodean sebuah string dengan menggunakan kode Huffman dank ode ASCII. Sebuah string berisi 100.000 karakter dengan komposisi karakter 'a' sebanyak 45.000, karakter 'b' sebanyak 13.000, karakter 'c' sebanyak 12.000, karakter 'd' sebanyak 16.000, karakter 'e' sebanyak 9.000, dan karakter 'f' sebanyak 5.000 akan dikodekan dengan menggunakan *Huffman Coding* dan kode ASCII untuk kemudian dibandingkan hasilnya.

3.1.1. Kode ASCII

Pada gambar 5 ditunjukkan kode ASCII untuk keenam karakter pada contoh di atas.

Karakter	ASCII	Ukuran
a	01000001	8 bit
b	01000010	8 bit
c	01000011	8 bit
d	01000100	8 bit
e	01000101	8 bit
f	01000110	8 bit

Gambar 5 : Tabel kode ASCII untuk karakter a, b, c, d, e, f.

Berdasarkan tabel pada gambar 5, representasi string pada contoh di atas dalam kode ASCII akan menggunakan memori sebesar :

- untuk karakter ‘a’
  45.000 x 8 bit = 360.000 bit
- untuk karakter ‘b’
  13.000 x 8 bit = 104.000 bit
- untuk karakter ‘c’
  12.000 x 8 bit = 96.000 bit
- untuk karakter ‘d’
  16.000 x 8 bit = 128.000 bit
- untuk karakter ‘e’
  9.000 x 8 bit = 72.000 bit
- untuk karakter ‘f’
  5.000 x 8 bit = 40.000 bit

Jumlah bit yang diperlukan adalah sebesar : 360.000 bit + 104.000 bit + 96.000 bit + 128.000 bit + 72.000 bit + 40.000 bit = 800.000 bit atau setara dengan 100.000 byte (8 bit = 1 byte).

3.1.2. Kode Huffman (*Huffman Coding*)

Pada gambar 6 ditunjukkan tabel kode Huffman untuk keenam karakter pada contoh diatas.

Karakter	Kekerapan	Peluang	Kode Huffman	Ukuran
a	45.000	0.45	0	1 bit
b	13.000	0.13	101	3 bit
c	12.000	0.12	100	3 bit
d	16.000	0.16	111	3 bit
e	9.000	0.09	1101	4 bit
f	5.000	0.05	1100	4 bit

Gambar 6 : Tabel kode Huffman untuk contoh 3.1.2.

Berdasarkan tabel pada gambar 6, representasi string pada contoh di atas dalam kode Huffman akan menggunakan memori sebesar :

- untuk karakter ‘a’
  45.000 x 1 bit = 45.000 bit
- untuk karakter ‘b’
  13.000 x 3 bit = 39.000 bit
- untuk karakter ‘c’
  12.000 x 3 bit = 36.000 bit
- untuk karakter ‘d’
  16.000 x 3 bit = 54.000 bit
- untuk karakter ‘e’
  9.000 x 4 bit = 36.000 bit
- untuk karakter ‘f’
  5.000 x 4 bit = 20.000 bit

Jumlah bit yang diperlukan adalah sebesar : 45.000 bit + 39.000 bit + 36.000 bit + 54.000 bit + 36.000 bit + 20.000 bit = 224.000 bit atau setara dengan 28000 byte.

3.1.3 Perbandingan keefektifan kode Huffman dan kode ASCII

Pada upabab 3.1.1. dan upabab 3.1.2. pengkodean dua buah string yang sama memberikan hasil pemakaian memori yang berbeda. Dengan menggunakan kode ASCII memori yang dipakai adalah sebesar 100.000 byte sedangkan dengan menggunakan kode Huffman memori yang dipakai adalah sebesar 28.000 byte. Hal ini berarti *Huffman Coding* dapat mengompress data hingga lebih dari 70% dibandingkan terhadap kode ASCII !

Hal ini menunjukkan kode Huffman merupakan salah satu metode pengkodean (*encoding*) yang cukup efektif baik dalam segi efektivitas maupun dalam segi keakuratan data.

3.2. Realisasi *Huffman Coding* dalam dunia nyata

Aplikasi *Huffman Coding* dalam dunia nyata sangat luas. Huffman Coding ini tidak terbatas hanya pada pengkodean data berupa string (seperti yang dicontohkan) tetapi juga dapat berlaku pada berbagai jenis data lain. Salah satu contohnya adalah pada CD/DVD yang menggunakan pixel hitam dan pixel putih dalam menyimpan datanya. Pixel hitam dan putih ini dapat digambarkan sebagai bit 1 dan bit 0. Sehingga *Huffman Coding* dapat juga diterapkan dalam penyimpanan data menggunakan media CD/DVD.

4. KESIMPULAN

Huffman Coding merupakan salah satu algoritma yang baik yang dapat digunakan untuk mengkodekan data. Hal ini dikarenakan *Huffman Coding* dapat dipergunakan pada berbagai jenis data dan pengaplikasiannya yang cukup mudah dengan memberikan hasil yang efektif.

Dalam pengkodean data (*Encoding*) menggunakan *Huffman Coding* dibutuhkan langkah – langkah :

- Pembentukan pohon Huffman
- Pengkodean data berdasarkan pohon Huffman

Dalam penguraian kode (*Decoding*) sebuah pesan

yang dikodekan (enkripsi) menggunakan *Huffman Coding*, dibutuhkan langkah – langkah :

1. Baca hasil enkripsi
2. dekripsi sesuai dengan pohon Huffman

Notasi O (Big – O) dari Algoritma Huffman adalah O(n + d log d). O(d log d) untuk membangun pohon Huffman. O(n) untuk membangun tabel kode Huffman dan pengkodean (enkripsi).

Huffman Coding dapat mengompres data mulai dari 22 % hingga 91 % tanpa adanya data yang hilang[3].

5. UCAPAN TERIMA KASIH

Makalah ini dibuat untuk memenuhi tugas mata kuliah Matematika Diskrit. Penulis mengucapkan terima kasih kepada dosen pengajar dan pembimbing mata kuliah Matematika Diskrit yang diikuti oleh penulis. Tanpa ilmu yang diajarkan kepada penulis, tidak mungkin penulis mampu menulis makalah ini.

Penulis juga memberikan ucapan terima kasih kepada setiap pihak yang telah membantu penulis dalam memperoleh data untuk menulis makalah ini. Terutama kepada setiap institusi pendidikan yang telah bersedia memberikan sarana dalam internet untuk memberikan data yang berhubungan dengan makalah ini.

Penulis juga memberikan ucapan terima kasih kepada setiap orang yang telah bersedia memberikan hasil tulisannya secara gratis untuk dapat penulis pakai dalam menulis makalah ini.

Ucapan terima kasih ini penulis berikan karena tanpa bantuan mereka semua, penulis tidak mungkin dapat menyelesaikan makalah ini.

DAFTAR REFERENSI

[1] *Huffman Coding* http://en.wikipedia.org/wiki/Huffman_code.

[2] Rinaldi Munir, 2003, Diktat Kuliah Matematika Diskrit, Penerbit ITB.

[3] McIntyre, David, 1985, Data compression using static Huffman code-decode tables. New York : ACM.

[4] *Huffman Coding* http://motivate.maths.org