Penggunaan Pohon Huffman Sebagai Sarana Kompresi *Lossless Data*

Aditya Rizkiadi Chernadi - 13506049
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
If16049@students.if.itb.ac.id

Abstraksi— Algoritma Huffman (Huffman Coding) adalah sebuah algoritma entropi encoding yang digunakan pada kompresi lossless data. Kompresi lossless data adalah sebuah kelas algoritma kompresi data yang memungkinkan data orisinil direkonstruksi secara utuh dari data terkompresinya. Teknik Huffman Coding digunakan dengan membuat pohon biner, lalu dilanjutkan ke tahap encoding dan diselesaikan pada tahap decoding. Prinsip utama dari algoritma ini adalah encoding yang dilakukan terhadap karakter yang memiliki frekuensi cukup tinggi dengan serangkaian bit yang lebih panjang.

Index Terms— Huffman Coding, string, biner, pohon, kode, prefiks.

I. PENDAHULUAN

File secara fisik terbentuk dari kumpulan karakter yang menjadi suatu kesatuan. File yang memuat banyak karakter yang beragam dapat menimbulkan problematika pada media penyimpanan dan kecepatan transmisi data.

Media penyimpanan yang terbatas, menekan semua orang untuk menemukan sebuah cara yang dapat digunakan untuk melakukan kompresi terhadap file. Saat ini terdapat banyak algoritma untuk melakukan kompresi data, seperti *Arithmetic, Block-Sorting Lossless, Dynamic Markov Compression*, LIFO, LZHUF, PPM, ataupun Shannon-Fano, namun algoritma-algoritma tersebut memiliki kekurangan dalam hal kemudahan implementasinya dibanding algoritma Huffman.

Kompresi data adalah proses *encoding* informasi menggunakan lebih sedikit bit dibanding representasi awalnya yang belum ter-*encode*, dengan menggunakan skema *encoding* yang spesifik.

Kompresi berguna untuk membantu mengurangi konsumsi sumber daya berharga seperti *hard disk* ataupun *bandwidth*.

Tujuan dari penulisan makalah ini adalah untuk menyelidiki efektivitas Huffman Coding dalam kompresi lossless data. Langkah-langkah yang dilakukan penulis untuk mencapai tujuan tersebut adalah studi literatur dari referensi-referensi yang tersedia di internet.

II. PEMBENTUKAN POHON HUFFMAN

Huffman Coding menggunakan metode spesifik untuk memilih representasi untuk masing-masing simbol, dan menghasilkan kode prefiks, yang mengekspresikan simbol yang paling banyak muncul dengan *string bits* yang lebih pendek.

Kode prefiks adalah himpunan yang berisi sekumpulan kode biner, dimana pada kode prefik ini tidak ada kode biner yang menjadi awal bagi kode biner yang lain. Kode prefiks biasanya direpresentasikan sebagai pohon biner yang diberikan nilai atau label. Untuk cabang kiri pada pohon biner diberi label 0, sedangkan pada cabang kanan pada pohon biner diberi label 1. Rangkaian bit yang terbentuk pada setiap lintasan dari akar ke daun merupakan kode prefiks untuk karakter yang berpadanan. Pohon biner ini biasa disebut pohon Huffman.

Langkah-langkah pembentukan pohon Huffman adalah sebagai berikut :

1. Baca semua karakter di dalam teks untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter penyusun teks dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul di-assign dengan frekuensi kemunculan karakter tersebut.

2. Terapkan strategi algoritma greedy sebagai berikut : Gabungkan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah akar. Setelah digabungkan akar tersebut akan mempunyaifrekuensi yang merupakan jumlah dari frekuensi dua buah pohon-pohon penyusunnya.

3. Ulangi langkah 2 sampai hanya tersisa satu buah pohon Huffman. Agar pemilihan dua pohon yang akan digabungkan berlangsung cepat, maka semua yang ada selalu terurut menaik berdasarkan frekuensi.

Sebagai contoh, dalam kode ASCII string 7 huruf "ABACCDA", dembutuhkan representasi 7 × 8 bit = 56 bit (7 byte = 56 bit).

Dimisalkan kode ASCII untuk keempat karakter tersebut adalah sebagai berikut :

Tabel 1. Kode ASCII

Simbol	Kode ASCII
A	01000001
B	01000010
C	01000011
D	01000100

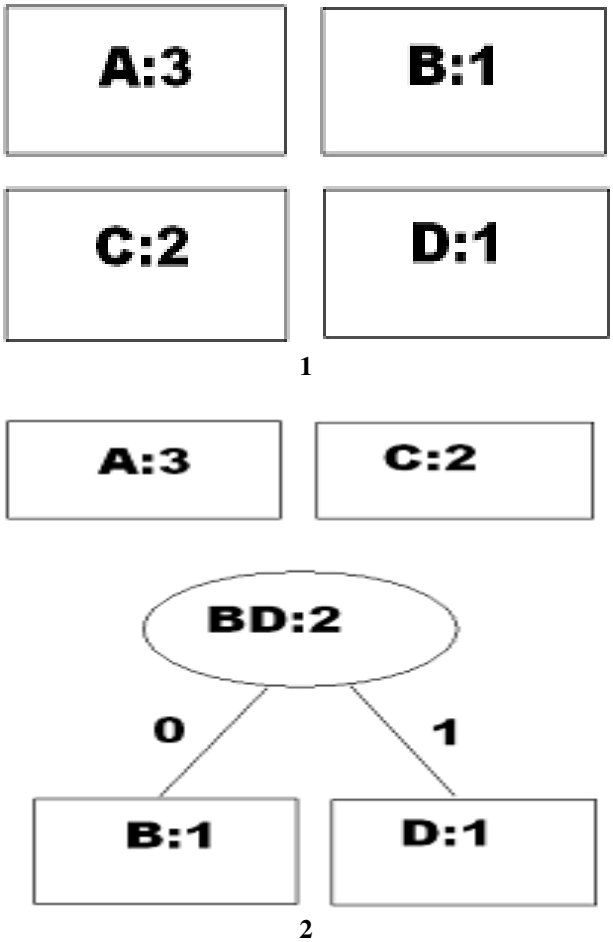
Dengan mengikuti ketentuan kode ASCII diatas, representasi rangkaian bit dari string "ABACCDA" :
01000001010000100100000101000011010000110100010001000001

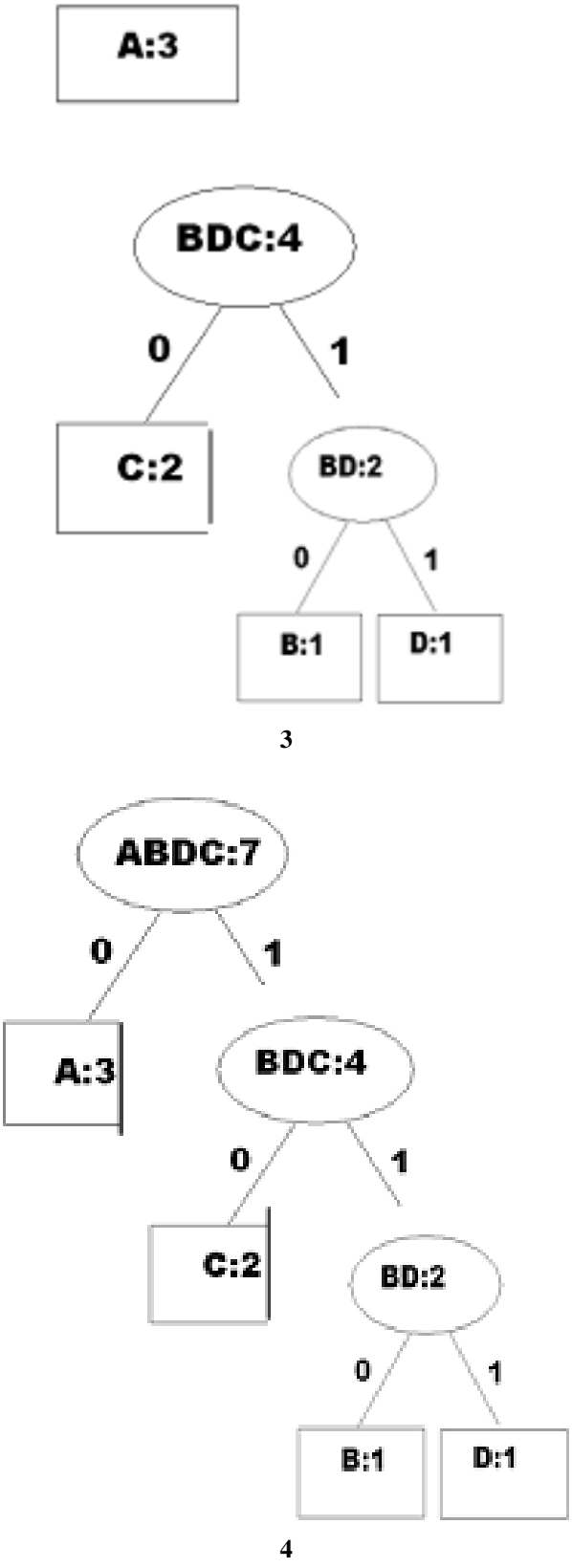
Seperti yang dapat kita lihat, string yang terdiri hanya dari 7 karakter saja representasinya sudah sepanjang 56 bit. Pada pesan "ABACCDA", kekerapan kemunculan A adalah 3, kekerapan B adalah 1, kekerapan C adalah 2, dan kekerapan D adalah 1.

Tabel 2. Tabel Kekerapan dan Peluang pada string "ABACCDA"

Simbol	Kekerapan	Peluang
A	3	3/7
B	1	1/7
C	2	2/7
D	1	1/7

Berikut ini ilustrasi pembentukan pohon huffmannya :





Gambar 1. Ilustrasi Pembentukan Pohon Huffman string "ABACCDA"

III. PROSES *ENCODING*

Encoding adalah cara menyusun string biner dari string yang ada. Proses *encoding* untuk satu karakter dimulai dengan membuat pohon Huffman dari string terlebih dahulu. Setelah itu, kode untuk satu karakter dibuat dengan menyusun nama string biner yang dibaca dari akar sampai ke daun pohon Huffman yang dibuat.

Berikut ini adalah langkah-langkah untuk meng-*encoding* suatu string biner :

1. Tentukan karakter yang akan di-*encoding*,
2. Mulai dari akar, baca setiap bit yang ada pada cabang yang bersesuaian sampai ketemu daun dimana karakter itu berada,
3. Ulangi langkah 2 sampai seluruh karakter selsesai di-*encoding*

Berikut ini hasil encoding dari string "ABACCDA"

Tabel 3. Kode Huffman Hasil *Encoding* string "ABACCDA"

Simbol	Kode Huffman
A	0
B	110
C	10
D	111

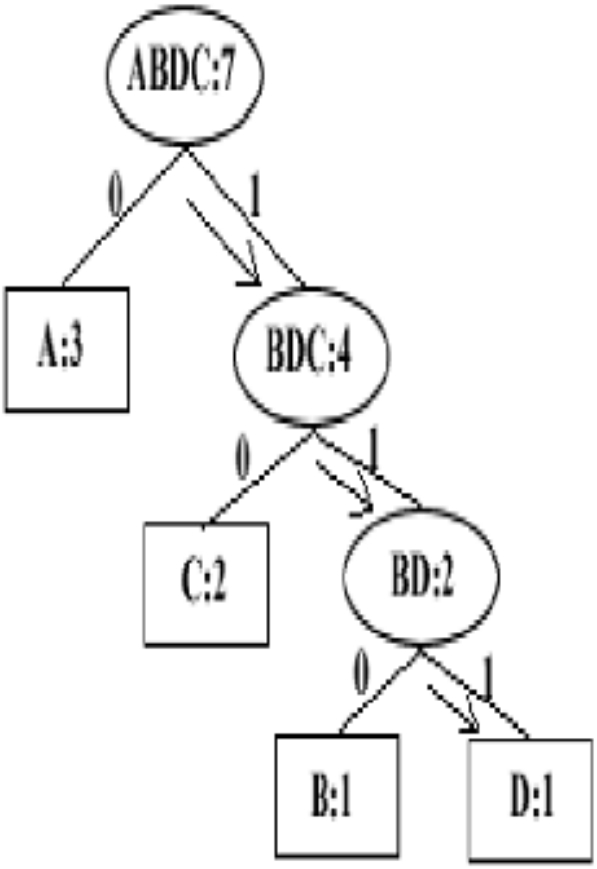
Dari hasil tersebut, representasi biner string "ABACCDA" menjadi : 011001010111.

IV. PROSES *DECODING*

Decoding merupakan kebalikan dari *encoding*. Sehingga dapat diartikan bahwa *decoding* adalah penyusunan kembali data dari string biner menjadi sebuah karakter. *Decoding* dapat dilakukan dengan dua cara, yang pertama dengan menggunakan pohon Huffman dan yang kedua dengan menggunakan tabel kode Huffman.

Berikut ini adalah langkah-langkah yang dilakukan untuk men-*decoding* suatu string biner dengan menggunakan pohon Huffman:

1. Dimulai dari akar, baca sebuah bit dari string biner.
2. Untuk setiap bit pada langkah 1, lakukan traversal pada cabang yang bersesuaian.
3. Ulangi langkah 1, 2 dan 3 sampai bertemu daun. Kodekan rangkaian bit yang telah dibaca dengan karakter di daun.
4. Ulangi dari langkah 1 sampai semua bit di dalam string habis.



Gambar 2. Ilustrasi Proses *Decoding* string "ABACCDA"

Cara yang kedua adalah dengan menggunakan tabel kode Huffman. Sebagai contoh kita akan menggunakan kode Huffman pada Tabel 1 untuk merepresentasikan string "ABACCDA".

Dengan menggunakan Tabel 1 string tersebut akan direpresentasikan menjadi rangkaian bit : 0 110 0 10 10 1110. Jadi, jumlah bit yang dibutuhkan hanya 13 bit. Dari Tabel 1 tampak bahwa kode untuk sebuah simbol/karakter tidak boleh menjadi awalan dari kode simbol yang lain guna menghindari keraguan (ambiguitas) dalam proses dekompresi atau decoding. Karena tiap kode Huffman yang dihasilkan unik, maka proses decoding dapat dilakukan dengan mudah.

Contoh: saat membaca kode bit pertama dalam rangkaian bit "011001010110", yaitu bit "0", dapat langsung disimpulkan bahwa kode bit "0" merupakan pemetaan dari simbol "A". Kemudian baca kode bit selanjutnya, yaitu bit "1". Tidak ada kode Huffman "1", lalu baca kode bit selanjutnya, sehingga menjadi "11". Tidak ada juga kode Huffman "11", lalu baca lagi kode bit berikutnya, sehingga menjadi "110". Rangkaian kode bit "110" adalah pemetaan dari simbol "B".

V. KOMPLEKSITAS ALGORITMA HUFFMAN

Algoritma Huffman mempunyai kompleksitas waktu O(n

log n), karena dalam melakukan satu kali iterasi pada saat penggabungan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah akar membutuhkan waktu O(log n), dan proses itu dilakukan berkali-kali hingga tersisa satu buah pohon Huffman saja, sehingga proses tersebut dilakukan sebanyak n kali.

VI. PENGUJIAN HUFFMAN CODING

Pada pengujian digunakan, kita akan meng-*encoding* sebuah teks yang berisi 100.000 string, diantaranya 45.000 karakter 'g', 13.000 karakter 'o', 12.000 karakter 'p', 16.000 karakter 'h', 9.000 karakter 'e', dan 5.000 karakter 'r' dengan menggunakan 3 cara, yaitu dengan menggunakan kode ASCII , kode 3-bit dan kode Huffman. Setelah itu ketiga kode tersebut akan dibandingkan satu sama lainnya.

Dengan kode ASCII :

Tabel 4. Kode ASCII "gopher"

Karakter	ASCII	Biner
G	103	1100111
O	111	1101111
P	112	1110000
H	104	1101000
E	101	1100101
R	114	1110010

Untuk meng-*encoding* teks tersebut kita membutuhkan sebanyak:

- untuk karakter 'g'
  45.000 x 8 bit (1100111) = 360.000 bit
- untuk karakter 'o'
  13.000 x 8bit (1101111) = 104.000 bit
- untuk karakter 'p'
  12.000 x 8bit (1110000) = 96.000 bit
- untuk karakter 'h'
  16.000 x 8bit (1101 000) = 128.000 bit
- untuk karakter 'e'
  9.000 x 8bit (1100101) = 72.000 bit
- untuk karakter 'r'
  5.000 x 8bit (1110010) = 40.000 bit

Jumlah bit yang digunakan Kode ASCII untuk karakter "g,o,p,h,e,r," = 360.000 + 104.000 + 96.000 + 128.000 + 72.000 + 40.000 = 800.000 bit

Dengan Kode 3-bit :

Tabel 5. Kode 3-bit "gopher"

Karakter	Kode	Biner
g	0	000
o	1	001
p	2	010
h	3	011
e	4	100
r	5	101

Untuk meng-encoding teks tersebut kita membutuhkan sebanyak:

- untuk karakter 'g'
  45.000 x 3 bit (000) = 135.000 bit
- untuk karakter 'o'
  13.000 x 3bit (001) = 39.000 bit
- untuk karakter 'p'
  12.000 x 3bit (010) = 36.000 bit
- untuk karakter 'h'
  16.000 x 3bit (011) = 48.000 bit
- untuk karakter 'e'
  9.000 x 3bit (100) = 27.000 bit
- untuk karakter 'r'
  5.000 x 3bit (101) = 15.000 bit

Jumlah bit yang digunakan 3-bit kode untuk karakter "g,o,p,h,e,r," = 135.000 + 39.000 + 36.000 + 48.000 + 27.000 + 15.000 = 300.000 bit.

Dengan Huffman Coding:

Karakter	Kekerapan	Peluang	Kode Huffman
g	45.000	45/100	0
o	13.000	13/100	101
p	12.000	12/100	100
h	16.000	16/100	111
e	9.000	9/100	1101
r	5.000	5/100	1100

Untuk meng-*encoding* teks tersebut kita membutuhkan sebanyak:

- untuk karakter 'g'
  45.000 x 1 bit (0) = 45.000 bit
- untuk karakter 'o'
  13.000 x 3bit (101) = 39.000 bit
- untuk karakter 'p'
  12.000 x 3bit (110) = 36.000 bit
- untuk karakter 'h'
  16.000 x 3bit (111) = 48.000 bit
- untuk karakter 'e'

9.000 x 4bit (1101) = 36.000 bit

- untuk karakter ‘r’
  5.000 x 4bit (1100) = 20.000 bit

Jumlah bit yang digunakan kode huffman untuk karakter “g,o,p,h,e,r” = 45.000 + 39.000 + 36.000 + 48.000 + 36.000 + 20.000 = 224.000 bit

Dari data diatas kita dapat lihat bahwa dengan menggunakan kode ASCII untuk meng-*encoding* teks tersebut membutuhkan 800.000 bit, sedangkan dengan menggunakan 3-bit
kode dibutuhkan 300.000 bit dan dengan menggunakan kode Huffman hanya membutuhkan 224.000. Jadi, untuk kasus tersebut algoritma huffman dapat mengompres teks sebesar 70% dibandingkan kita menggunakan kode ASCII dan sebesar 25,3% dibandingkan kita menggunakan 3-bit kode.

VII. KESIMPULAN

1. Algoritma Huffman adalah salah satu algoritma kompresi, yang banyak digunakan dalam kompresi teks.
2. Terdapat 3 tahapan dalam menggunakan algoritma Huffman, yaitu:
   - membentuk pohon Huffman,
   - melakukan *encoding* dengan menggunakan pohon Huffman,
   - melakukan *decoding*
3. Algoritma Huffman mempunyai kompleksitas waktu O(n log n).
4. Algoritma Huffman adalah salah satu algoritma yang menggunakan prinsip algoritma greedy dalam penyusunan pohon Huffman
5. Dari hasil pengujian yang dilakukan, algoritma Huffman dapat mengompres teks sekitar 70% jika dibandingkan dengan menggunakan kode ASCII dan sekitar 25% jika dibandingkan dengan kita menggunakan 3-bit kode.

VIII. DAFTAR REFERENSI

[1] Huffman Coding.
http://en.wikipedia.org/wiki/Huffman_coding.
Tanggal akses : 10 Desember 2010.

[2] Rinaldi Munir, 2006. *Diktat Kuliah IF215 Matematika Diskrit*. Penerbit ITB.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2010

ttd

Aditya Rizkiadi Chernadi - 13506049