METODE POHON BINER HUFFMAN UNTUK KOMPRESI DATA STRING KARAKTER

Muqtafi Akhmad (13508059)

Teknik Informatika ITB
Bandung
e-mail: if18059@students.if.itb.ac.id

ABSTRAK

Dalam makalah ini akan dibahas tentang pemanfaatan metode pohon biner Huffman untuk melakukan kompresi data berupa string yang terdiri dari karakter. Pembahasan dimulai dengan pengenalan pohon dan pohon biner serta beberapa terminologi pohon. Kemudian akan dijelaskan sedikit pengantar mengenai metode pohon biner Huffman dan metode Huffman dasar serta metode Huffman menggunakan Canonical Huffman Code sebagai salah satu variasi dari metode Huffman. Setelah mengetahui dasar teori mengenai metode kompresi Huffman, kemudian kita akan mencoba melakukan simulasi kompresi menggunakan metode Huffman dasar dan metode Huffman menggunaakan Canonical Huffman Code kemudian membandingkan hasil kompresi keduanya sehingga kita dapat membandingkan keduanya.

Kata kunci: pohon biner, metode Huffman, akar, anak, kode Huffman

1. PENDAHULUAN

Dalam kode ASCII, setiap karakter direpresentasikan dengan 8 bit, atau 1 byte. Berarti untuk menyimpan informasi satu karakter dibutuhkan 1 byte memori. Dengan demikian untuk menyimpan sebuah teks dengan jumlah karakter besar, diperlukan memori yang besar pula dengan perbandingan yang linier terhadap jumlah karakter. Salah satu solusi untuk mengatasi hal ini adalah dengan menggunakan metode Huffman.

Metode Huffman ditemukan oleh seorang mahasiswa MIT bernama David Huffman. Algoritma ini bekerja dengan ide pengodean karakter dengan kemunculan tinggi dengan bit yang sedikit dan karakter dengan kemunculan rendah dikodekan dengan bit yang lebih besar. Dengan demikian, ukuran memori untuk menyimpan informasi dapat diminimalisasikan tanpa ada informasi yang terbuang (lossless). Yang akan kita bahas adalah metode yang dasar dan salah satu variasi dari metode Huffman, yaitu dengan menggunakan Canonical Huffman Code.

2. DASAR TEORI

2.1 Pohon

2.1.1 Pengertian dan Terminologi Pohon

Pohon adalah graf tak berarah terhubung yang tidak mengandung sirkuit. Misalkan G = (V, E) adalah graf tak berarah sederhana dan jumlah simpulnya n. Maka semua pernyataan di bawah ini adalah ekuivalen :

1. G adalah sebuah pohon
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal
3. G terhubung dan memiliki m = n-1 buah sisi
4. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit
5. G terhubung dan semua sisinya adalah jembatan (jembatan adalah sisi yang apabila dihapus graf akan terpecah menjadi dua komponen)

Pohon berakar adalah pohon yang sebuah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah yang menjauh dari akar. Untuk selanjutnya pohon berakar ini disebut pohon. Beberapa terminologi dalam pohon berakar :

1. Anak(child) dan Orangtua(parent)

Misalkan X adalah sebuah simpul dalam pohon berakar. Simpul Y dikatakan anak simpul X jika ada sisi dari simpul X ke Y. Dalam hal demikian, X disebut sebagai orangtua dari Y.

2. Lintasan (path)

Lintasan dari simpul v1 ke simpul vk adalah runtunan simpul-simpul v1, v2, ...vk sedemikian sehingga vi adalah orangtua dari v(i+1).

3. Keturunan (descendant) dan Leluhur(ancestor)

Jika terdapat lintasan dari simpul X ke simpul Y di dalam pohon, maka X adalah leluhur simpul Y dan Y adalh keturunan simpul X.

4. Saudara kandung (siblings)

Dua simpul dikatakan bersaudara apabila memiliki orangtua yang sama.

5. Upapohon(subtree)

Misalkan X adalah sebuah simpul di dalam pohon T. Yang dimaksud dengan upapohon dengan X sebagai akarnya adalah upagraf T' = (V', E') sedemikian sehingga V' mengandung X dan semua keeturunannya dan E' mengandung sisi-sisi dalam semua lintasan yang berasal dari X.

6. Derajat(degree)

Derajat sebuah simpul adalah jumlah upapohon (atau jumlah anak) pada simpul tersebut. Derajat sebuah pohon adalah derajat maksimum dari simpul-simpulnya.

7. Daun(leaf)

Daun adalah simpul berderajat nol.

8. Simpul dalam(internal node)

Simpul dalam adalah simpul yang mempunyai anak.

9. Aras(level)

Akar memiliki aras 0, sedangkan aras simpul lainnya adalah 1 + panjang lintasan dari akar ke simpul tersebut.

10. Tinggi(height) atau Kedalaman(depth)

Tinggi atau kedalaman adalah aras maksimum dari sebuah pohon.

2.1.2 Pohon Biner

Pohon berakar yang setiap simpulnya memiliki paling banyak n buah anak disebut pohon n-ary. Jika n = 2 maka disebut pohon biner. Dalam pohon biner dikenal istilah anak kanan(right child) dan anak kiri(left child).

Beberapa terminologi dalam pohon biner :

1. Upapohon kiri(left subtree)

pohon yang akarnya adalah anak kiri

2. Upapohon kanan(right subtree)

pohon yang akarnya adalah upapohon kanan

3. Pohon condong kiri(skew left)

pohon yang semua simpulnya terletak di kiri saja

4. Pohon condong kanan(skew right)

pohon yang semua simpulnya terletak di kanan saja

2.2 Pohon Huffman

Kode Hufman adalah string biner yang digunakan untuk mengkodekan setiap karakter di dalam data. Ide dari kode Huffman adalah menggunakan representasi dengan bit yang sesedikit mungkin untuk karakter yang memiliki jumlah kemunculan yang tinggi, dan sebaliknya, karakter yang memiliki jumlah kemunculan lebih sedikit memiliki representasi dengan jumlah bit yang lebih sedikit.

Kode Huffman pada dasarnya merupakan kode prefiks (prefix code). Kode prefiks adalah himpunan yang berisi sekumpulan kode biner yang dalam hal ini tidak ada kode biner yang menjadi awal bagi kode biner yang lain.

Kode prefiks biasanya direpresentasikan sebagai pohon biner yang berlabel, dimana setiap sisi diberi label 0 atau 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.

2.3 Pohon Huffman Dengan Teknik Dasar

Pertama baca string karakter lalu mendatanya ke dalam larik dengan ukuran sesuai dengan jumlah simbol yang digunakan, n dan memberikan anggapan awal bahwa semua node adalah daun yang berisi karakter itu sendiri dan bobotnya (jumlah kemunculannya dalam string) dan sebuah pranala ke parent (opsional) dari node untuk mempermudah pembacaan kode (dalam decoding, dimulai dari daun kiri). Node internal memiliki informasi karakter, bobot, pranala ke dua anak dan ke parent (opsional). Sebagai konvensi, bit '0' merepresentasikan anak sebelah kiri dan bit '1' merepresentasikan anak sebelah kanan. Sebuah pohon yang sudah terbentuk memiliki n daun dan n-1 buah node internal

Proses dimulai dengan node daun yang berisi kemungkinan kemunculan karakter, kemudian sebuah node baru dengan dua anak dengan dua kemungkinan kemunculan terkecil dibentuk dengan kemungkinan merupakan jumlah dari kemungkinan anaknya. Kemudian anggap node baru ini sebuah daun kemudian cari node lain yang memiliki kemungkinan kemunculan terkecil kemudian lakukan prosedur yang sama. Ulangi prosedur ini hingga semua daun habis.

Salah satu cara konstruksi pohon biner Huffman yang paling mudah adalah menggunakan priority queue di mana node dengan kemungkinan kemunculan terkecil memiliki proiritas terbesar

1. Buat sebuah node daun untuk masing-masing katakter dan menambahkannya ke dalam priority queue.
2. Apabila ada lebih dari satu node dalam queue :
    1. Hapus dua node dengan prioritas tertinggi (kemungkinan kemunculan terkecil) dari queue.
    2. Bentuk sebuah node internal baru dengan dua node tadi sebagai anak dengan kemungkinan merupakan jumlah dari kemungkinan kemunculan dua anaknya.
    3. Tambahkan node baru tersebut ke dalam priority queue. Ulangi hingga tertinggal satu node, node tersebut adalah akar dari pohon Huffman.

Karena efisiensi dari struktur data priority queue membutuhkan O(log n) per penyisipan (penambahan elemen dalam queue), dan sebuah pohon dengan n buah daun memiliki 2n-1 node, maka algoritma ini memiliki kompleksitas operasi O (n log n).

2.3.1 Proses Encoding

Proses encoding menggunakan teknik dasar ini dilakukan dengan me-replace karakter pada string dengan kode Huffman yang berkaitan, maka akan didapatkan string dengan kode Huffman.

2.3.2 Proses Decoding

Untuk proses decoding ini, kita membtuhkan sebuah pohon biner Huffman yang sudah kita bentuk untuk string berisi kode Huffman yang akan kita deCode. Dalam implementasi kita akan menggunakan sebuah pointer ke pohon biner yang diinisialisasi dengan alamat akarnya. Kemudian telusuri anak-anak dari setiap node internal menggunakan fungsi rekursif dengan parameter alamat ke pohon biner dan mengembalikan karakter yang dicari. Berikut adalah algoritmanya (menggunakan pendekatan bahasa C) :

```
char Huffman_deCode(pointer to binary tree node)

{

 /*pertama cek apakah pointer sekarang menunjuk sebuah node internal atau daun. Pengecekan dilakukan dengan mengecek bobot dari node*/

if (IsDaun(node))

 {

 /*sekarang ada di dalam node daun, berisi karakter*/

 return (node->info);

 }

else

/*sekarang kita ada dalam node internal*/

 {

 if (get_bit() == 0)

 /*bit selanjutnya adalah 0*/

 {

return (Huffman_deCode(node->left_node));

 }

 else

 /*bit selanjutnya adalah 1*/

 {

 return (Huffman_deCode(node->right_node));

 }

}
```

Proses ini dipakai setiap kali ada masih ada bit di dalam string bit hasil encoding dengan pohon Huffman.

2.4 Pohon Huffman Menggunakan Canonical Huffman Code

2.4.1 Proses Encoding
Aturan-aturan proses encoding pada algoritma Huffman Kanonik adalah sebagai berikut:

1. Panjang kode untuk suatu simpul adalah sebesar aras+1 simpul tersebut.
2. String biner simpul paling dalam yang terletak paling kiri diberi nilai 0 semuanya. Untuk simpul berikutnya(bergeser dari kiri ke kanan) string binernya naik satu nilai dari simpul sebelumnya.
3. Apabila semua simpul pada kedalaman yang sama telah di-enCode, maka proses encoding dilanjutkan ke aras yang lebih rendah. Hanya saja string binernya tidak dimulai dengan semuanya 0. String biner simpul paling kiri dimulai dengan kode baru. Kode baru itu merupakan kenaikan 1 nilai dari string biner simpul yang terakhir di-enCode namun biner paling belakangnya dihilangkan sehingga panjang kodenya berkurang satu. Simpul berikutnya di-enCode dengan string binernya naik satu nilai(bergeser dari kiri ke kanan).
4. Proses berhenti bila telah mencapai akar.

Setelah terbentuk sebuah pohon Huffman, maka yang perlu dilakukan adalah membentuk tabel kode Huffman, dilakukan dengan sebuah fungsi

```
void make_Codes(current_node, depth, Code)
{
/*cek apakah sebuah simbol */
if( IsDaun(current_node ) )
 {
 /*merupakan daun, maka akar berisi simbol, kemudian membuat kode dan panjang kode pada tabel kode Huffman*/
    /*"current_node->value" adalah sebuat bit untuk tabel kode Huffman*/
 Huffman_Code_table[current_node->value].Code=Code;
 Huffman_Code_table[current_node-
```

```
>value].Codelength=depth;
 return;  /*akhir rekursi */
 }
else
 {
 /*sekarang terdapat di dalam node internal, update kode dan melakukan rekursi. Pertama anak kiri, kemudian lakukan pergeseran kode ke kiri. Kemudian tambahkan bit '0'. Kita akan melakukan rekursi ke anak, maka kedalaman ditambah*/
depth++;
    /*rekursi untuk anak kiri */
 make_Codes(current_node->left_node,depth,Code);
    /*Sekarang kita melakukan rekursi terhadap anak kemudian kita menambahkan sebuah bit '1' di posisi paling kanan kode menggunakan 'OR'*/
    Code = Code OR 1;
    /*rekursi ke anak kanan*/
 make_Codes(current_node->right_node,depth,Code);
 }
}
```

Tabel kode Huffman ini akan digunakan ketika kita melakukan decoding.

2.4.2 Proses Decoding

Proses decoding memiliki beberapa cara alternatif.

1. Alternatif pertama adalah dengan membuat tabel yang berisi karakter yang di-deCode dan jumlah bit yang digunakan. Alternatif pertama ini cocok digunakan pada data-data dengan panjang kode maksimum yang pendek karena ukuran tabel sebanding dengan panjang kode.
2. Alternatif kedua adalah membuat tabel untuk setiap panjang kode. Lalu tabel yang sesuai akan dicari untuk setiap input yang dimasukkan (tabel ini adalah tabel yang dibentuk menggunakan algoritma yang ada pada proses encoding).
3. Alternatif ketiga adalah dengan membuat multilevel tabel. Pertama-tama dicek beberapa bit kode pertama, lalu tabel yang akan dipakai diberitahu. Kemudian pada tabel berikutnya akan dicari karakter yang sesuai berdasarkan panjang kodenya.

3. ANALISIS

Dalam analisis ini kita akan mencoba melakukan encoding sebuah string karakter menggunakan teknik dasar dan menggunakan Canonical Huffman Code kemudian membandingkan efisiensi antara keduanya. Misalkan kita memiliki sebuah data berupa sebuah sting karakter berikut

kita janjian makan bakso nanti siang

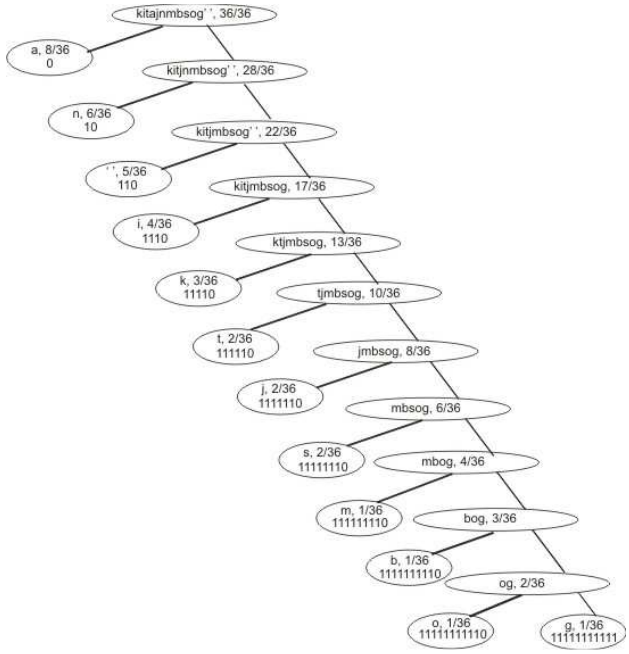
Kemudian melakukan satu kali pembacaan untuk mengitung kemunculan dari masing-masing karakter:

Tabel 1 jumlah kemunculan masing-masing karakter dalam string

Karakter	Jumlah
k	3
i	4
t	2
a	8
j	2
n	6
m	1
b	1
s	2
o	1
g	1
' ' (spasi)	5
Total karakter	36

1. Menggunakan teknik dasar

Pertama kita membangun sebuah pohon Huffman menggunakan cara biasa



Gambar 1. Pohon biner Huffman didapatkan menggunakan teknik dasar

Kemudian coba ubah string asal menjadi kode dalam bit Huffman yang sudah didapat dari pohon Huffman di atas:

111101110111110011011111100101111111011100101101
111111100111100101101111111111001111011111110111
111111101101001011111101110110111111101110010111
111111111

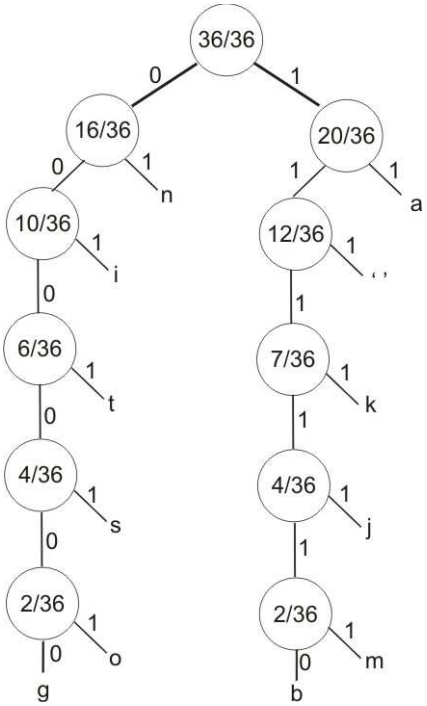
Dengan jumlah bit : 149

Setelah mengetahui jumlah bit hasil kompresi, kita bandingkan rasio antara hasil kompresi dengan jumlah bit asal

$$rasio = \frac{149}{288} = 51{,}73\%$$

2. Menggunakan Canonical Huffman Code

Menggunakan string yang sama, kita akan membangun sebuah pohon Huffman menggunakan Canonical Huffman Code.



Gambar 2. Pohon biner Huffman didapatkan menggunakan Canonical Huffman Code

Kemudian mencoba untuk mengubah string asal ke kode bit Huffman yang baru saja kita dapatkan :

11110010001111111111111011111100111011111111111111111111011111111101011110000100000111101110100100100000110011101000000

Dengan jumlah bit : 119

Sama seperti sebelumnya, kita bandingkan rasio antara hasil kompresi dengan jumlah bit asal

$$rasio = \frac{119}{288} = 41{,}32\%$$

4. KESIMPULAN

1. Algoritma Huffman merupakan slah satu algoritma kompresi teks yang cukup efisien, dengan rasio minimal berkisar pada 50% (dalam analisis didapatkan rasio yang melebihi 50% karena banyaknya karakter dengan jumlah kemunculan yang sama dan kita memberikan kode Huffman yang berbeda untuk masing-masing karakter tersebut).
2. Jika dibandingkan, algoritma Huffman menggunakan Canonical Huffman Code menghasilkan hasil kompresi yang lebih efisien daripada metode menggunaakn teknik dasar. Hal ini dapat diamati dari bentuk pohon kode Huffman untuk setiap metode. Pada pohon hasil pengodean menggunakan metode teknik dasar, setiap karakter memiliki jumlah representasi bit yang sama (kecuali dua karakter dengan jumlah kemunculan yang paling sedikit), perbedaan jumlah bit untuk masing-masing karakter ini juga menyebabkan tingginya pohon yang terbentuk (dalam contoh yang dianalisis pohon yang terbentuk memiliki 11 aras). Sedangkan, pada pohon hasil pengodean menggunakan Canonical Huffman dapat dilihat setiap karakter pada level yang sama direpresentasikan dengan jumlah bit yang sama, sehingga memang ada karakter-karakter yang direpresentasikan dengan jumlah bit yang sama tapi tetap unik. Dengan adanya karakter yang direpresentasikan dengan jumlah bit sama ini pohon yang terbentuk pun ‘lebih lebar’ dan memiliki aras yang lebih rendah (dalam contoh ada 6 aras).

5.REFERENSI

[1] Wikipedia (2009), http://en.wikipedia.org/wiki/Canonical_Huffman_code (18 Desember 2009, pukul 08:34)

[2] Wikipedia (2009), http://en.wikipedia.org/wiki/Adaptive_Huffman_coding (18 Desember 2009, pukul 08:45)

[3] Wikipedia (2009), http://en.wikipedia.org/wiki/Greedy_algorithm (19 Desember 2009, pukul 15:25)

[4] Rinaldi Munir, “Matemtika Diskrit”, Penerbit Informatika Bandung, Cetakan Ketiga (Maret 2009).