Pemampatan Data dengan Algoritma Huffman Kanonik

Yavta Mabaklini Ginting[1], Bemby Bantara Narendra[2]

Laboratorium Ilmu dan Rekayasa Komputasi
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if13101@students.if.itb.ac.id[1], if13105@students.if.itb.ac.id[2]

Abstrak

Kode Huffman adalah salah satu algoritma pemampatan data yang pernah dibahas dalam kuliah Matematika Diskrit pokok bahasan Pohon (aplikasi dari pohon biner). Teknik pemampatan data dengan algoritma Huffman mampu memberikan penghematan sampai sebesar tiga puluh persen. Namun algoritma ini masih dirasa kurang efisien karena sulitnya proses *decoding* atau proses menyusun data kembali dari string biner pada *encoding*. Untuk itu kami akan membahas tentang algoritma Huffman Kanonik jauh lebih baik daripada algoritma Huffman biasa. Pembentukan pohon Huffman pada algoritma Huffman Kanonik hampir sama dengan pembentukan pohon Huffman pada algoritma Huffman biasa. Bedanya adalah pada algoritma Huffman Kanonik, pohon Huffman-nya tidak diberi label 0 atau 1 pada sisi-sisinya. Sehingga algoritma Huffman Kanonik memberikan keuntungan seperti *decoding* yang lebih cepat dan algoritma ini dapat merekonstruksi kode dengan hanya mengetahui panjang *string* biner karakter yang bersangkutan.

***Kata kunci**: huffman codes, canonical huffman codes, huffman encoding, huffman decoding, huffman coding*

1. Pendahuluan

Data yang berukuran besar seringkali sulit untuk disimpan dalam media penyimpanan yang berukuran terbatas. Karena alasan itu, data yang besar sebelum disimpan biasanya dimampatkan terlebih dahulu agar ukurannya lebih kecil dari semula.

Salah satu algoritma yang sering digunakan dalam teknik pemampatan data adalah kode huffman. Teknik ini mampu memampatkan sampai dengan tiga puluh persen dari ukuran semula. Tetapi proses *decoding string* biner menjadi data kembali masih kurang efisien. Karena itu, kami mencoba membahas salah satu algoritma yang mirip dengan algoritma Huffman tetapi lebih efisien yaitu algoritma Huffman Kanonik.

2. Algoritma Huffman

2.1. Algoritma Huffman Biasa

2.1.1. Pembentukan Pohon Huffman

Kode Hufman adalah *string* biner yang digunakan untuk mengkodekan setiap karakter di dalam data. Prinsip yang digunakan pada kode Huffman adalah karakter yang paling sering muncul di dalam data (data dapat berupa pesan yang dikirim atau data yang disimpan di dalam *storage*) dikodekan (*encode*) dengan kode yang lebih pendek, sedangkan karakter yang relatif jarang muncul dikodekan dengan kode yang lebih panjang.

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 (cabang kiri) atau 1 (cabang kanan). 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 data untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter penyusun data dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul di-*assign* dengan frekuensi kemunculan karakter tersebut.
2. Terapkan strategi *greedy* sebagai berikut : gabungkan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah

akar. Akar mempunyai frekuensi yang merupakan jumlah dari frekuensi dua buah pohon penyusunnya.

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

2.1.2. Proses Encoding

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

2.1.3. Proses Decoding

Decoding merupakan kebalikan dari *encoding*. *Decoding* berarti menyusun data dari *string* biner.
Langkah-langkah men-*decoding* suatu *string* biner adalah sebagai berikut :

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

2.2. Algoritma Huffman Kanonik

2.2.1. Pembentukan Pohon Huffman

Pembentukan pohon Huffman pada algoritma Huffman Kanonik hampir sama dengan pembentukan pohon Huffman pada algoritma Huffman biasa. Bedanya adalah pada algoritma Huffman Kanonik, pohon Huffman-nya tidak diberi label 0 atau 1 pada sisi-sisinya.

2.2.2. 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.

Untuk lebih jelasnya, perhatikan contoh berikut.
Pada contoh ini, kode prefiks tidak direpresentasikan sebagai pohon biner yang berlabel, tetapi dalam sintaks kurung (*bracket syntax*). Setiap kurung berisi subpohon yang ditulis dari kiri ke kanan. Misalnya, ((a,b),c) menyatakan pohon biner dimana anak kiri dari akar adalah subpohon yang berisi ‘a’ sebagai anak kiri dan ‘b’ sebagai anak kanan serta anak kanan dari akar adalah ‘c’.

Contoh:
Terdapat pohon Huffman sebagai berikut:
((((B,F),A),E),(((G,C),D),H)) dengan kedalaman 4.

Karakter	*String* Biner Huffman Biasa	*String* Biner Huffman Kanonik	Aras
A	001	010	2
B	0000	0000	3
C	1001	0001	3
D	101	011	2
E	01	10	1
F	0001	0010	3
G	1000	0011	3
H	11	11	1

Pada pohon Huffman tersebut, karakter ‘B’ terletak pada simpul paling dalam dan paling kiri dan diberi kode 0000. Karakter ‘C’ terletak di sebelah kanan ‘B’ pada aras yang sama sehingga diberi kode 0001 yang lebih satu nilai dari 0000. Lalu karakter ‘F’ dan ‘G’ diberi kode 0010 dan 0011 dengan cara yang sama. Untuk karakter ‘A’ yang berada pada aras yang lebih rendah diberi kode yang lebih satu nilai dari 0011 yaitu 0100 namun biner belakangnya dihilangkan menjadi 010. Karakter ‘D’ yang berada pada aras yang sama dengan ‘A’ diberi kode 011 yang lebih satu nilai dari 010. Demikian seterusnya hingga semua karakter telah di-*encode*.

2.2.3. Proses Decoding

Proses *decoding* memiliki beberapa cara alternatif. 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.

Alternatif kedua adalah membuat tabel untuk setiap panjang kode. Lalu tabel yang sesuai akan dicari untuk setiap input yang dimasukkan.

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. Kesimpulan

Algoritma Huffman Kanonik memiliki beberapa keuntungan. Proses *decoding* algoritma ini efisien karena menggunakan tabel untuk penyusunan data dari *string* biner. Daftar karakter dan kode biner tersusun dengan rapi pada tabel. Dengan begitu, penyusunan data dari *string* biner dapat dengan cepat dilakukan.

Keuntungan lainnya adalah algoritma ini memungkinkan penyusunan karakter dengan hanya mengetahui panjang *string* biner karakter yang bersangkutan. Tiap karakter pada pohon Huffman diberikan kode *string* biner yang panjangnya sesuai dengan aras karakter tersebut. Dengan memperhatikan aras dan urutan karakter dari kiri ke kanan, maka dapat diketahui *string* biner suatu karakter. Dengan begitu, *string* biner dapat disusun menjadi karakter.

Daftar Pustaka

[1] Rinaldi Munir, 2005, *Diktat Kuliah IF2251 Strategi Algoritmik*, Penerbit ITB.

[2] Practical Huffman Coding http://www.compressconsult.com/huffman/ diakses tanggal 15 Mei 2005 pukul 11.00 WIB

[3] Data Structures and Algorithms: Introduction http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/introduction.html diakses tanggal 15 Mei 2005 pukul 11.00 WB