Pemampatan dengan Menggunakan Algoritma Huffman Dinamik : Algoritma FGK dan Algoritma Vitter

Chandra Sutikno Oemaryadi
Jurusan Teknik Informatika ITB, Bandung 40116, email: if16075@students.if.itb.ac.id

Abstrak – Algoritma Huffman Statik adalah salah satu algoritma paling tua dan paling terkenal dalam pemampatan data. Untuk memampatkan data, algoritma Huffman statik memerlukan dua buah fase, yaitu pembentukan kode awalan dan pengkodean data berdasarkan kode awalan yang telah terbentuk. Adanya dua buah fase, mengakibatkan cukup lamanya waktu dan bit yang diperlukan untuk memampatkan data.

Pengembangan dari algoritma Huffman Statik adalah algoritma Huffman Dinamik, yang hanya memerlukan satu buah fase. Algoritma Huffman Statik pertama kali ditemukan oleh Faller dan Gallager, selanjutnya dikembangkan oleh Knuth, menghasilkan algoritma yang dikenal sebagai algoritma FGK. Versi terakhir dari algoritma Huffman Dinamik adalah algoritma Vitter, yang diuraikan oleh Jeffrey Scott Vitter. Prinsip yang digunakan dalam algoritma Huffman Dinamik adalah pengkodean secara dinamik dan sibling property. Implementasi dari metode ini antara lain algoritma FGK dan algoritma Vitter. Di akhir makalah ini, akah dibahas perbandingan hasil kompresi dengan menggunakan algoritma FGK dan algoritma Vitter.

Kata Kunci : algoritma Huffman Statik, algoritma Huffman Dinamik, algoritma FGK, algoritma Vitter

1. PENDAHULUAN

Algoritma Huffman Statik adalah algoritma yang dibuat oleh seorang mahasiswa MIT, David A. Huffman, dan dipublikasikan pada tahun 1952 dalam paper “A Method for the Construction of Minimum-Redundancy Codes”. Huffman menjadi anggota fakultas MIT begitu ia lulus, dan di kemudian hari menjadi salah satu pendiri Departemen Ilmu Komputer Universitas California, Santa Cruz, yang sekarang menjadi bagian dari Baskin School of Engineering. Algoritma Huffman Statik menggunakan metode tertentu untuk menghasilkan kode awalan sebagai representasi setiap simbol yang terdapat di dalam data.

Pengembangan dari algoritma Huffman statik adalah algoritma Huffman Dinamik. Algoritma Huffman Dinamik pertama kali ditemukan oleh Faller dan Gallager, selanjutnya dikembangkan oleh Knuth, menghasilkan algoritma yang dikenal sebagai algoritma FGK. Versi terakhir dari algoritma Huffman Dinamik diuraikan oleh Jeffrey Scott Vitter, dan dikenal sebagai algoritma Vitter.

2. DASAR TEORI

Definisi lambang yang akan digunakan dalam bagian “Dasar Teori”

a_j = karakter ke-j
j = banyak karakter yang sudah diproses
k = banyak jenis karakter yang sudah diproses
w_j = kemunculan karakter a_j yang sudah diproses
l_j = lintasan dari akar ke daun a_j

2.1 Algoritma Huffman Statik

Algoritma Huffman Statik adalah salah satu algoritma yang paling tua dan paling terkenal dalam pemampatan data. Prinsip Algoritma Huffman Statik adalah mengkodekan setiap simbol dengan rangkaian bit 0 dan 1, di mana simbol yang memilki frekuensi lebih sedikit memiliki rangkaian bit yang lebih panjang daripada simbol yang memiliki frekuensi banyak dalam data. Kode yang dihasilkan adalah kode awalan, yaitu himpunan kode sedemikian sehingga tidak ada anggota kumpulan yang merupakan awalan dari anggota yang lain. Misalnya, himpunan A{00,010,0111} adalah kode awalan, namun himpunan B{00,010,001} bukan kode awalan, karena ‘00’ adalah prefiks dari ‘001’.

Algoritma Huffman Statik memiliki dua buah fasa, yaitu fasa pembentukan kode awalan dan pengkodean data berdasarkan kode awalan yang telah terbentuk.

Kode awalan dibentuk dengan menggunakan pohon biner. Simpul yang terdapat dalam pohon biner dapat berupa simpul daun dan simpul dalam. Pertama-tama, setiap simpul adalah simpul daun, yang menyimpan informasi simpul yang disimpan dan bobot simpul, yaitu frekuensi timpul tersebut di dalam data. Simpul dalam menyimpan informasi link ke dua buah anak dan bobot simpul, yaitu jumlah bobot kedua simpul anaknya. Sesuai konvensi yang umum dipakai, bit ‘0’ merepresentasikan lintasan anak kiri, dan bit ‘1’ merepresentasikan lintasan anak kanan. Pohon yang sudah selesai dikerjakan akan memiliki n buah simpul daun dan (n-1) buah simpul dalam, dan disebut dengan pohon Huffman.

Salah satu metode untuk membentuk pohon Huffman adalah dengan menggunakan list of node L. List L adalah list terurut menaik berdasarkan bobot simpul.

Pseudo Code :

```
Menyimpan k simpul di list L;
while (banyak node yang terdapat di L lebih dari sama dengan 2) do    {
                    Mengambil 2 buah simpul dengan bobot terkecil, x dan y;
                    Membuat simpul baru p dan menjadikan p simpul orang tua dari x dan y;
Bobot p=bobot x + bobot y;
                    Masukkan p ke dalam L sesuai urutan bobot;
end;
}
```

Simpul terakhir yang terdapat di L di akhir algoritma adalah akar dari pohon Huffman yang diinginkan. Setiap kali melakukan iterasi di dalam kalang, ada kemungkinan untuk memilih lebih dari dua buah simpul dengan bobot minimum yang akan diambil dari list. Pemilihan simpul yang berbeda akan menghasilkan struktur pohon Huffman yang berbeda, namun setiap kemungkinan pohon Huffman yang terbentuk akan memiliki kedalaman yang sama.

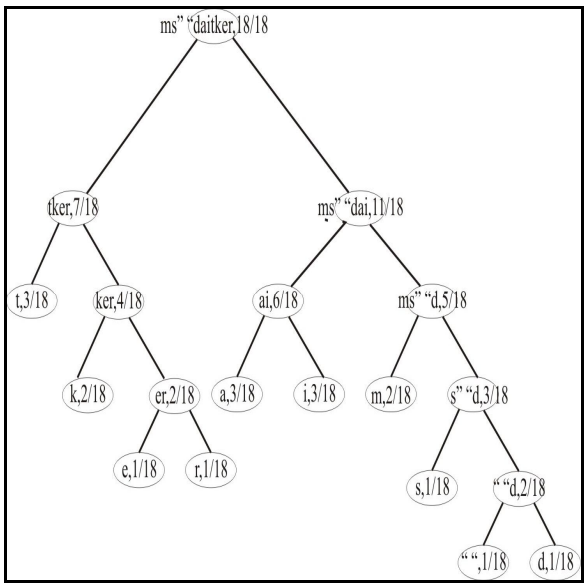
Fasa kedua dari algoritma Huffman Statik adalah mengkodekan data dengan menggunakan pohon Huffman yang telah terbentuk dalam fasa 1. Setiap simbol a_j dikodekan dengan bit '0' dan '1', sesuai dengan lintasan dari akar pohon Huffman ke simpul a_j, menggunakan konvensi umum bahwa '0' berarti 'ke kiri' dan '1' berarti 'ke kanan'.

Contoh :
Pemampatan teks "matematika diskrit" dengan menggunakan algoritma Huffman
Tabel kekerapan :

Simbol	Kekerapan	Peluang
" "	1	1/18
d	1	1/18
e	1	1/18
r	1	1/18
s	1	1/18
k	2	2/18
m	2	2/18
a	3	3/18
i	3	3/18
t	3	3/18

Tabel 1 – tabel kekerapan dan peluang dari setiap karakter yang terdapat dalam teks "matematika diskrit"

Pohon Huffman yang terbentuk dari algoritma Huffman Statik (pseudo code) untuk teks "matematika diskrit"



Gambar 1 – pohon Huffman yang dibentuk oleh algoritma Huffman Statik untuk teks "matematika diskrit"

Simbol	Kode Huffman
t	00
k	010
e	0110
r	0111
a	100
i	101
m	110
s	1110
" "	11110
d	11111

Tabel 2 – tabel kode Huffman yang terbentuk berdasarkan pohon Huffman pada Gambar 1

Dari tabel, kita dapat melihat bahwa simbol dengan frekuensi kemunculan sedikit (misalnya "d", yang memiliki frekuensi 1, dikodekan dengan 11111) memiliki rangkaian bit lebih panjang daripada simbol dengan frekuensi kemunculan banyak (misalnya t, yang memiliki frekuensi 3, dikodekan dengan 00). Kita juga dapat melihat bahwa kode Huffman yang terbentuk adalah kode awalan.

Dengan menggunakan kode Huffman, teks "matematika diskrit" direpresentasikan menjadi rangkaian bit :

11010000011011010000101010100111101111111011
110010011110100

Jadi, dengan menggunakan kode Huffman, jumlah bit yang dibutuhkan untuk merepresentasikan teks "matematika diskrit" adalah 58 bit. Hal ini membuktikan bahwa penggunaan kode Huffman jauh lebih efisien daripada penggunaan kode ASCII, yang akan membutuhkan 18x8 bit untuk merepresentasikan teks "matematika diskrit".

2.2 Algoritma Huffman Dinamik

Algoritma Huffman Statik memiliki sebuah kekurangan yang cukup fatal, yaitu memerlukan dua fasa dalam proses pemampatan data, fasa satu untuk menghitung frekuensi tiap karakter dan membentuk pohon Huffman, dan fase kedua untuk memampatkan data dengan memanfaatkan fohon Huffman. Hal ini mengakibatkan bertambahnya waktu dan bit yang diperlukan untuk melakukan pemampatan data.

Berbeda dengan algoritma Huffman statik yang membutuhkan dua fase, algoritma Huffman Dinamik hanya membutuhkan satu fase dalam memampatkan data, dimana proses penghitungan frekuensi karakter dan pembuatan pohon Huffman identik di encoder dan decoder dilakukan secara dinamik pada saat membaca data. Pembuatan pohon Huffman secara dinamik sangat efektif jika data yang akan datang belum dapat dipastikan kedatangannya, misalnya pada audio dan vieo streaming.

Basis dari algoritma Huffman Dinamik adalah *sibling property*, yang didefinisikan oleh Galager. Pohon biner memiliki *sibling property* jika dan hanya jika :

1) setiap simpul, kecuali akar memiliki sudara kandung
2) untuk setiap simpul di dalam pohon, bobot akar ≥ bobot simpul kanan ≥ bobot simpul kiri
3) urutan setiap simbol berbanding lurus terhadap bobot simbol tersebut, semakin besar urutan maka akan semakin besar frekuensi simbol di dalam data yang sudah dibaca, demikian juga sebaliknya
4) simpul-simpul yang semakin dekat dengan akar memiliki bobot yang semakin besar
5) bobot simpul dalam adalah jumlah bobot simpul kanan dan bobot simpul kiri

Gallager juga membuktikan bahwa setiap pohon adalah pohon Huffman jika dan hanya jika pohon tersebut memiliki *sibling property*.

Daun-daun dari pohon Huffman menyatakan simbol yang terdapat dalam data, dan bobot dari tiap daun menyatakan frekuensi simbol tersebut di dalam data. Pohon yang dibangun oleh algoritma Huffman Dinamik harus memiliki *sibling property*.

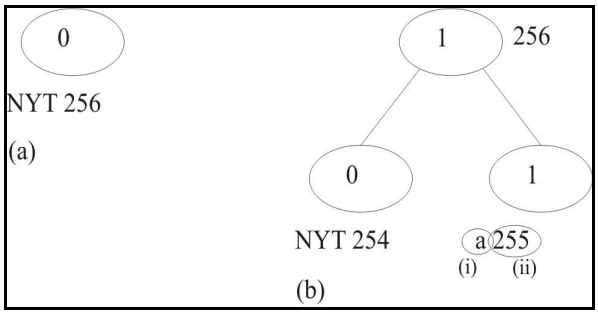
Implementasi dari algoritma Huffman Dinamik antara lain algoritma FGK dan algoritma Vitter. Algoritma FGK bekerja cukup baik dalam memampatkan data, dan cukup mudah untuk diimplementasikan, namun tidak sebaik algoritma Vitter dalam meminimalisir ketinggian dari pohon Huffman guna mempersingkat waktu yang dibutuhkan untuk mengakses daun. Algoritma Vitter dapat meminimalisir ketinggian dari pohon Huffman, namun sulit untuk diimplementasikan untuk menghasilkan program yang bekerja dengan cepat. Algoritma FGK dan algoritma Vitter akan dibahas secara lebih mendalam pada bagian berikutnya.

2.3 Algoritma FGK dan Algoritma Vitter

Basis dari kedua algoritma ini adalah *sibling property*, seperti yang sudah dijelaskan pada bagian "Algoritma Huffman Dinamik". Selain menyimpan informasi bobot, simbol (untuk simpul daun), dan link anak (untuk simpul dalam), setiap simpul yang terdapat di dalam pohon Huffman juga menyimpan sebuah informasi tambahan, yaitu angka urut. Angka urut ini sangat diperlukan keberadaannya dalam algoritma Vitter.

Angka urut adalah angka unik yang dimiliki oleh setiap simpul di dalam pohon, dalam arti tidak lebih dari satu simpul yang memiliki angka urut tersebut. Simpul-simpul dengan ketinggian yang sama akan memiliki angka urut terurut membesar dari kiri ke kanan. Angka urut diberikan kepada simpul ketika simpul tersebut pertama kali dibuat. Angka urut yang diberikan adalah angka urut terbesar yang belum digunakan oleh simpul-simpul yang lain.

Pohon Huffman juga memiliki sebuah simpul khusus, yang disebut simpul-0 pada algoritma FGK, dan simpul NYT(Not Yet Transmitted) pada algoritma Vitter. Ketika sebuah simbol yang belum terdapat di dalam pohon Huffman dikirimkan/dibaca, simpul NYT akan membentuk dua buah anak, anak kanan menyimpan informasi simbol baru tersebut, dan anak kiri menjadi simpul NYT yang baru. Anak kanan mempunyai angka urut lebih besar daripada anak kiri. Misal, jika angka urut terbesar yang belum digunakan adalah x, maka anak kanan mempunyai angka urut x, sedangkan anak kiri mempunyai angka urut (x-1). Pohon Huffman yang pertama kali terbentuk hanya memiliki sebuah simpul, yaitu simpul NYT.



Gambar 2 – contoh simpul dan pohon Huffman (a) Pohon Huffman mula-mula, hanya terdiri dari satu buah simpul NYT (b) Pohon Huffman setelah memproses karakter "a" (i) Simbol yang direpresentasikan oleh simpul daun (ii) Nomor urut simpul

Berikut akan dijelaskan mengenai teori dasar dan prinsip yang digunakan dalam algoritma Vitter.

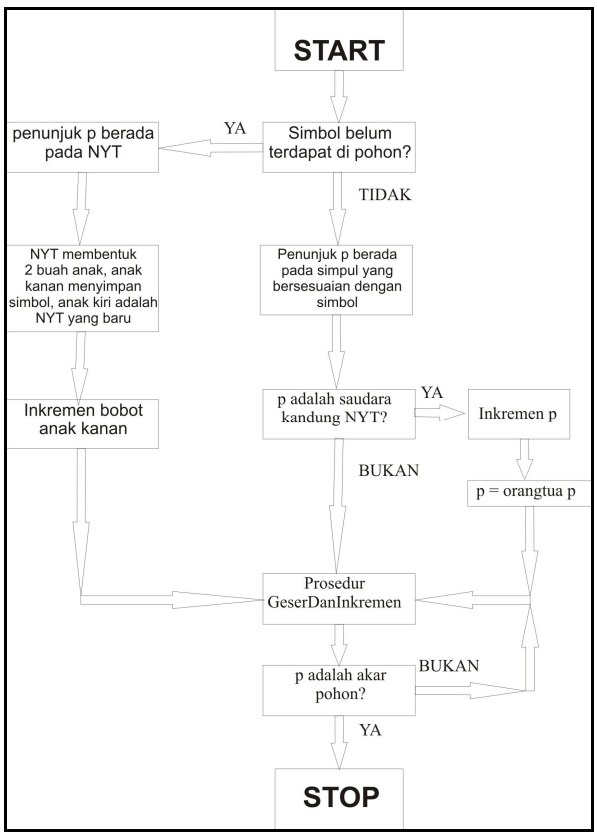
Algoritma FGK memiliki sedikit perbedaan dengan algoritma Vitter, dan akan dijelaskan kemudian.

2.3.1 Algoritma Vitter

Pohon Huffman yang dimanipulasi oleh algoritma Vitter harus memenuhi syarat-syarat berikut ini :

1) Setiap simpul memiliki saudara kandung
2) Simpul dengan bobot lebih besar memiliki angka urut yang lebih besar daripada simpul dengan bobot lebih kecil
3) Dalam setiap ketinggian, simpul paling kanan akan memiliki angka urut terbesar, meskipun ada kemungkinan simpul lain yang memiliki bobot sama dengan simpul tersebut
4) Simpul daun menyimpan informasi nilai simbol, kecuali simpul NYT
5) Bobot simpul dalam adalah jumlah bobot dari kedua anaknya
6) Simpul-simpul yang memiliki bobot sama disimpan dalam sebuah **blok**

Dalam syarat-syarat di atas terdapat sebuah "benda" yang tidak terdapat dalam algoritma Huffman Statik maupun algoritma FGK, yaitu **blok**. **Blok** adalah list terurut dari simpul-simpul yang memiliki bobot sama w. **Blok** diurutkan mengecil berdasar angka urut yang dimiliki simpul-simpul tersebut, simpul yang pertama adalah simpul dengan angka urut terbesar.



Gambar 3 – Prosedur manipulasi pohon Huffman dengan menggunakan algoritma Vitter

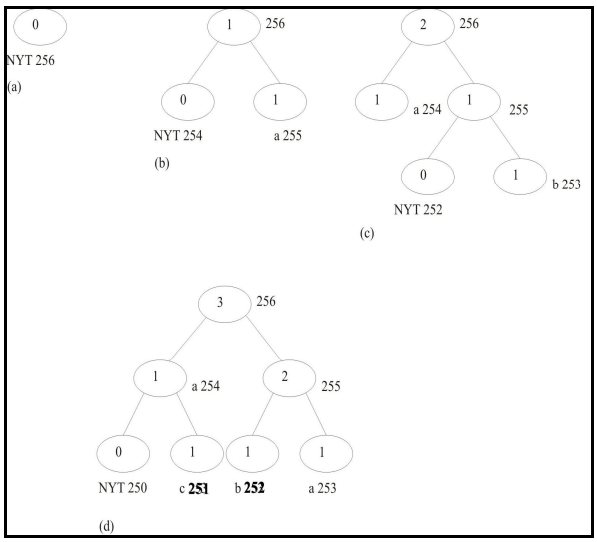
Pohon Huffman selalu memiliki sebuah akar dan simpul NYT, di mana simpul-NYT adalah simpul dengan angka urut terkecil di dalam pohon. Ketika sebuah simbol a_j diproses, program akan memeriksa apakah a_j terdapat di dalam k. Jika tidak, simpul NYT akan membentuk dua buah simpul anak, anak kanan menyimpan simbol tersebut, dan anak kiri menjadi simpul NYT yang baru. Jika a_j terdapat di dalam k, maka posisi a_j akan ditukar dengan posisi simpul pertama di blok w, lalu memanggil prosedur GeserDanInkremen atau menginkremen w_j. Hal ini bertujuan agar pohon yang terbentuk tetap memiliki sibling property.

Perubahan bobot dari simpul daun akibat penambahan simbol ke dalam pohon Huffman akan mempengaruhi simpul-simpul yang terdapat di atas simpul daun tersebut. Hal ini akan ditangani dengan prosedur GeserDanInkremen. Prosedur ini akan dilakukan berulang hingga penunjuk simpul menunjuk kepada akar dari pohon Huffman.

Ada beberapa hal yang perlu diperhatikan dalam menukar posisi simpul, antara lain :

- Akar dari pohon tidak boleh ditukar posisinya dengan apapun
- Tidak diperbolehkan untuk menukar posisi suatu simpul dengan simpul orang tua maupun simpul leluhurnya
- Angka urut dari dua buah simpul yang ditukar posisinya haruslah ditukar juga
- Jika p adalah saudara kandung dari simpul NYT, maka p diinkremen, tanpa perlu menukar posisi p dengan simpul pertama pada blok w_p

Prosedur GeserDanInkremen berfungsi untuk menempatkan suatu simpul p pada posisi pertama pada blok w_p dan menggeser posisi simpul-simpul tertentu pada blok w_p, sehingga ketika w_p diinkremen, pohon Huffman tetap memiliki sibling property. Ketika melakukan prosedur GeserDanInkremen, yang berubah adalah angka urut yang dimiliki simpul-simpul yang mengalami pergeseran. Misal, blok terdiri dari 2 buah simpul : a (angka urut 253) dan b (angka urut 251). Ketika sebuah simpul c (angka urut 245) menggeser blok, maka angka urut c menjadi 253, angka urut a menjadi 251, dan angka urut b menjadi 245.



Gambar 4 – pohon yang dihasilkan algoritma Vitter untuk teks "abc"

Keterangan Gambar (4) :

(a) Pohon Huffman mula-mula, hanya terdiri dari satu buah simpul NYT
(b) Pohon Huffman setelah menerima karakter "a", simpul NYT membentuk anak kanan yang menyimpan karakter "a", dan anak kiri yang menjadi simpul NYT yang baru
(c) Pohon Huffman menerima karakter "b", simpul NYT membentuk anak kanan yang menyimpan karakter "b", dan anak kiri yang menjadi simpul NYT yang baru. Penunjuk p berada pada simpul orang tua simpul "b". Memanggil prosedur GeserDanInkremen untuk p. Hasil prosedur, p berada pada posisi simpul "a", posisi simpul "a" digeser ke posisi p. W_p dan w_{akar} diinkremen.
(d) Pohon Huffman menerima karakter "c", simpul NYT membentuk anak kanan yang menyimpan karakter "c", dan anak kiri yang menjadi simpul NYT yang baru. Penunjuk p berada pada simpul orang tua simpul "c". Memanggil prosedur GeserDanInkremen untuk p. Hasil prosedur, p berada pada posisi simpul "a", posisi simpul "a" digeser ke posisi simpul "b", dan posisi simpul "b" digeser ke posisi simpul p mula-mula. W_p dan w_{akar} diinkremen.

Pseudo-code :

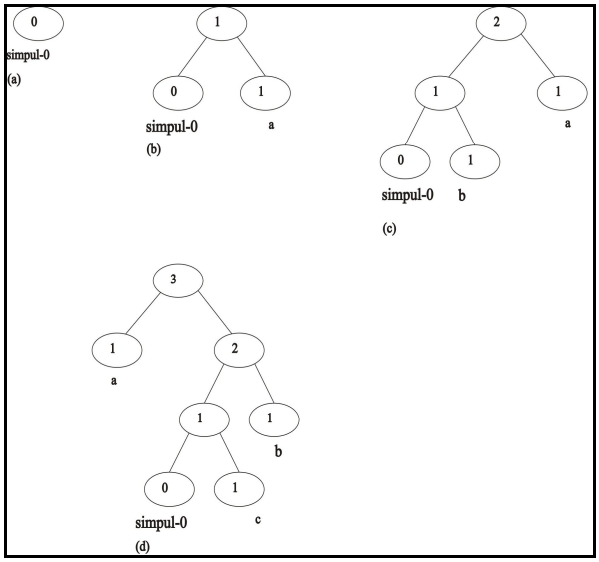
```
prosedur update  {
q=simpul daun bersesuaian dengan a(j+1);
if (q==NYT)      {
left(q)==NYT;
right(q)==a(t+1);
InkremenDaun(right(q));
}        else     {
         Menukar posisi q dengan simpul pertama di
         blok w(t+1);
         if (SaudaraKandung(q)==NYT)     {
         InkremenDaun(q);
         q=OrangTua(q);
         }
}
while (q!=Akar pohon Huffman)    {
GeserDanInkremen(q);
}
}

procedure GeserDanInkremen(p)    {
w=bobot p;
b=blok;
if(((p==simpul daun)&&(b==blok simpul-simpul dalam dengan bobot w))||((p==simpul dalam)&&(q==blok simpul-simpul daun dengan bobot w+1)))  {
         menggeser blok b, p adalah simpul pertama di b;
         wp=wp+1;
         if (p==simpul daun)        {
         p=orangtua p yang baru;
         }        else     {
         p=orangtua p yang lama;
         }
}
}
```

2.3.2 Algoritma FGK

Perbedaan mendasar dari algoritma FGK dan algoritma Vitter adalah algoritma FGK tidak memeriksa keseluruhan simpul di dalam pohon ketika mengubah bobot suatu simpul. Algoritma FGK hanya memeriksa saudara kandung nya, atau saudara kandung dari orang tuanya. Hal ini mengakibatkan pohon yang dihasilkan tidak memiliki tinggi minimum yang dapat, namun akan mempercepat proses pembaharuan pohon.

Proses algoritma FGK :

1) Memasukkan simbol a_j ke dalam pohon
2) Penunjuk p menunjuk simpul-0 atau simpul yang bersesuaian dengan a_{j+1}
3) Menginkremen bobot dari p
4) Memeriksa bobot dari saudara kandung p, tukar jika diperlukan, yaitu bobot saudara kandung di kanan lebih kecil daripada bobot p
5) Memerika bobot saudara kandung dari orang tua p, tukar jika diperlukan, yaitu saudara kandung dari orang tua tersebut terdapat di kanan dan memiliki bobot lebih kecil dari p
6) p=orang tua p
7) mengulangi (4), (5), (6) hingga p==akar



Gambar 5 – pohon yang dihasilkan algoritma FGK untuk teks "abc"

Keterangan gambar (5) :

(a) Pohon mula-mula, hanya terdiri dari satu buah simpul-0

(b) Pohon setelah menerima karakter "a", simpul-0 membentuk anak kanan yang menyimpan karakter "a", dan anak kiri yang menjadi simpul-0 yang baru

(c) Pohon setelah menerima karakter "b", simpul-0 membentuk anak kanan yang menyimpan karakter "b", dan anak kiri yang menjadi simpul NYT yang baru

(d) Pohon setelah menerima karakter "c", simpul-0 membentuk anak kanan yang menyimpan karakter "c", dan anak kiri yang menjadi simpul NYT yang baru. Simpul kakek dari "c" bertukar posisi dengan simpul "a", karena bobot simpul "a" lebih kecil dari bobot simpul kakek.

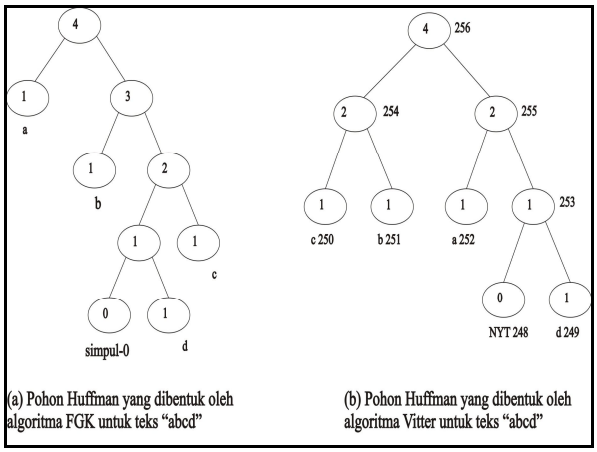
3. HASIL DAN PEMBAHASAN

Untuk melihat perbandingan hasil kompresi dengan menggunakan algoritma Vitter dan algoritma FGK, kita melihat ukuran data hasil kompresi dengan menggunakan kedua buah algoritma tersebut untuk teks "abcd".

Pohon Huffman yang dihasilkan oleh algoritma Vitter dan algoritma FGK dapat dilihat pada **gambar (6).** Kode dihasilkan berdasarkan lintasan yang ditempuh dari akar ke karakter yang bersesuaian. Sesuai konvensi yang umum dipakai, angka 0 ditambahkan ke rangkaian bit untuk setiap lintasan kiri yang diambil, angka 1 ditambahkan ke rangkaian bit untuk setiap lintasan kanan yang diambil.

Karakter	Kode Huffman algoritma Vitter	Kode Huffman algoritma FGK
a	10	0
b	01	10
c	00	111
d	111	1101

Tabel2 – Kode Huffman yang dihasilkan algoritma Vitter dan algoritma FGK untuk teks "abcd"



Gambar 6 – Pohon Huffman dihasilkan oleh algoritma FGK dan Vitter untuk teks "abcd"

Berdasarkan tabel2, maka teks "abcd" direpresentasikan menjadi rangkaian bit :

	abcd	Bit
Vitter	100100111	9
FGK	0101111101	10

Tabel3 – Hasil representasi teks "abcd" dengan menggunakan algoritma Vitter dan FGK

Dengan hasil ini, kita dapat melihat dengan jelas, bahwa jumlah bit yang dihasilkan oleh algoritma Vitter lebih sedikit daripada yang dihasilkan oleh algoritma FGK. Hal ini diakibatkan pohon yang dihasilkan algoritma FGK lebih tinggi daripada pohon yang dihasilkan oleh algoritma Vitter.

Bagaimana hubungan antara ketinggian pohon dan rangkaian bit yang dihasilkan? Semakin tinggi suatu pohon, maka akan semakin panjang lintasan yang diperlukan untuk mencapat simpul daun pada tingkat terdalam pohon tersebut. Semakin panjang lintasan, maka rangkaian bit yang dihasilkan akan semakin panjang, sehingga menambah jumlah bit yang dibutuhkan untuk merepresentasikan suatu teks tertentu.

Meskipun lebih baik dalam memampatkan data, algoritma Vitter cukup sulit untuk diimplementasikan dalam bahasa pemrograman. Algoritma Vitter juga memakan waktu yang cukup lama untuk melakukan pemampatan data, karena harus memeriksa keseluruhan pohon ketika akan melakukan penambahan bobot. Algoritma FGK bekerja lebih cepat daripada algoritma Vitter, karena hanya memeriksa simpul saudara kandung dan simpul saudara kandung dari orang tua.

4. KESIMPULAN

- Algoritma Huffman Statik lebih baik daripada Algoritma Huffman Dinamik, karena memiliki peta kode yang dapat berubah-ubah sesuai data yang sudah dibaca
- Algoritma Huffman Dinamik hanya memerlukan satu buah fasa dalam proses pemampatan data
- Basis dari algoritma Huffman Dinamik adalah sibling property
- Aplikasi dari Algoritma Huffman Dinamik antara lain algoritma FGK dan algoritma Vitter
- Algoritma FGK bekerja lebih cepat daripada algoritma Vitter, karena algoritma FGK hanya memeriksa simpul orangtua dan simpul paman (saudara kandung orangtua) dalam proses memperbaharui pohon
- Algoritma Vitter memeriksa seluruh simpul dalam proses memperbaharui pohon, sehingga bekerja lebih lambat daripada algoritma FGK, namun akan menghasilkan pohon Huffman dengan ketinggian minimum
- Pohon Huffman yang dihasilkan oleh algoritma Vitter lebih pendek daripada pohon Huffman yang dihasilkan algoritma FGK
- Ketinggian pohon menentukan jumlah bit yang diperlukan untuk merepresentasikan suatu teks, karena bit akan bertambah seiring bertambahnya lintasan yang harus ditempuh untuk mencapat suatu lintasan tertentu

5. DAFTAR REFERENSI

[1]Munir, Rinaldi. (2006). Diktat Kuliah IF2153 Matematika Diskrit Edisi Keempat. Program Studi Teknik Informatika, Sekolah Teknik Informatika dan Elektro, Institut Teknologi Bandung.

[2]Vitter, Jeffrey Scott. (1987) . *"Design and Analysis of Dynamic Huffman Codes*".(1987). http://www.cs.duke.edu~7Ejsv/Papers/catalog/node79.html
Tanggal akses : Selasa, 25 Desember pukul 07.52

[3]Wikipedia. Huffman coding. http://en.wikipedia.org/wiki/Huffman_coding
Tanggal akses : Sabtu, 22 Desember pukul 11.15

[4]Wikipedia. Adaptive Huffman Coding. http://en.wikipedia.org/wiki/Adaptive_Huffman_Coding
Tanggal akses : Senin, 24 Desember pukul 20.55

[5]Adaptive Huffman Coding. www.ics.uci.edu/~dan/pubs/DC-Sec4.html
Tanggal akses : Selasa, 25 Desember pukul 07.43

[6]Fayaz, Bharwani.(1999). Adaptive Huffman Coding. http://www.cs.mcgill.ca
Tanggal akses : Selasa, 25 Desember 2007 pukul 15.25

[7]Hendrawa. Huffman Coding. http://telecom.ee.itb.ac.id/~hend/ET5014/HuffmanCoding_07.ppt
Tanggal akses : Selasa, 25 Desember 2007 pukul 15.34

[8]Low, Jonathan.(2000). Adaptive Huffman Coding. http://cs.duke.edu
Tanggal akses : Selasa, 25 Desember pukul 07.42