PERBANDINGAN ALGORITMA HUFFMAN DENGAN ALGORITMA SHANNON-FANO

Gagarin Adhitama (13508089)
Program Studi Teknik Informatika
Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
Email : if18089@students.if.itb.ac.id

Abstrak *– Makalah ini membahas tentang perbandingan dua algoritma dengan proses kompresi. Yang pertama yaitu algoritma Shannon-fano dan algoritma Huffman. Pada intinya kedua algoritma ini sama, membentuk suatu kode yang memiliki redundansi minimum sehingga manajemen memori dapat dilakukan dengan baik. Pada dasarnya algoritma Huffman ini adalah pengembangan dari algortima Shannon-Fano, tapi keduanya memiliki perbedaan dalam cara kerjanya. Tepatnya adalah berbeda pada pembuatan pohon untuk melakukan pengkodean yang lebih ringkas dan efisien.*

Kata Kunci*: perbandingan, algortima Huffman, algoritma Shannon-Fano*

Abstract *- This paper discusses the comparison of two algorithms with the process of compression. The first is Shannon-Fano algorithm, and Huffman algorithms. In essence, these two algorithms are same, form a code that has a minimum redundancy so that managemen of memoryt can be done well. Huffman algorithm basically this is the development of the Shannon-Fano algorithm, but Huffman have differences in how it works. Especially in making tree to made a efficient code.*

Keywords*: comparison, algorithms Huffman, Shannon-Fano algorithm*

1. PENDAHULUAN

1.1 Pengertian Algoritma Huffman

Algoritma Huffman adalah salah satu algoritma kompresi tertua yang disusun oleh David Huffman pada tahun 1952. Algoritma tersebut digunakan untuk membuat kompresi jenis lossy compression, yaitu pemampatan data dimana tidak satu byte pun hilang sehingga data tersebut utuh dan disimpan sesuai dengan aslinya. Pada sejarahnya, Huffman sudah tidak dapat membuktikan apapun tentang kode apappun yang efisien, tapi ketika tugasnya hampir final, ia mendapatkan ide untuk menggunakan pohon binary untuk menyelesaikan masalahnya mencari kode yang efisien [3]. Pada dasarnya, algoritma Huffman ini bekerja seperti mesin sandi morse, dia membentuk suatu kode dari suatu karakter. Sehingga karakter tersebut memiliki rangkaian bit yang lebih pendek dibandingkan sebelumnya.

1.2 Pengertian Algoritma Shannon-Fano

Algoritma Shannon-Fano coding ditemukan oleh Claude Shannon (bapak teori informasi) dan Robert Fano pada tahun 1949. Pada saat itu metode ini merupakan metode yang paling baik tetapi hampir tidak pernah digunakan dan dikembangkan lagi setelah kemunculan algoritma Hufman. Pada dasarnya
metode ini menggantikan setiap simbol dengan sebuah alternatif kode biner yang panjangnya ditentukan berdasarkan probabilitas dari simbol tersebut Di bidang kompresi data, Shannon-Fano coding adalah teknik untuk membangun sebuah kode awalan didasarkan pada seperangkat simbol dan probabilitas (diperkirakan atau diukur). Namun, algoritmya ini dirasa kurang optimal dalam arti bahwa ia tidak mampu mencapai kode seefisien mungkin seperti kode diharapkan panjang seperti algoritma Huffman [2].

2. PEMBAHASAN

Pada dasarnya kedua algoritma ini menggunakan metode yang sama dalam membuat kode yang singkat. Pada awalnya, algoritma ini membuat sebuah pohon berupa simpul daun dan anak-anaknya yang memiliki probabilitas dari seringnya munculnya karakter tersebut dalam teks. Kemudian proses kedua yaitu proses encoding. Dari pohon tersebut, tiap karakternya akan memiliki identitas berupa bilangan biner untuk menyimpan memori. Proses pembentukan dari karakter menjadi biner inilah yang disebut dengan proses encoding. Untuk penjelasan bagaimana encoding itu akan diperjelas ketika pohon telah dibuat (pada butir selanjutnya). Proses yang terahir atau proses ketiga yaitu proses decoding. Decoding adalah proses kebalikan dari encoding, yaitu membalikan dari angaka-angka biner yang pendek (untuk meringkas memori) diubah lagi menjadi karakter yang panjang lagi.

2.1 Cara Kerja Algoritma Huffman

2.1.1 Pembuatan Pohon Huffman

Pohon Huffman ini dibentuk berdasarkan kode prefiks. Kode biner dibentuk secara prefiks dan kode biner ini tidak mungkin terbentuk sama satu sama lainnya. Karakter-karakter yang akan direpresentasikan dalam biner, dipisahkan ke dalam cabang pohon biner dan beri frekuensinya. Cabang sebelah kiri diberi bit 0 sebagai identitas, dan bit kanan diberi angka 1. pada akhirnya bit ini akan dibaca dari akar hingga simpul dari suatu karakter itu sehingga terbentuk angka biner identitas untuk meringkas memori sehingga menjadi efisien. Langkah dalam membentuk pohon Huffman ini dapat diringkas menjadi sebagai berikut :

1. hitung semua frekuensi kemunculan tiap karkternya. Hal ini dapat dilakukan hanya dengan menghitung (memroses) semua karakter dari awal hingga akhir dengan saru kali lewat (*one way pass).*
2. kemudian terapkan strategi algoritma greedy. Apa itu algoritma greedy? Yaitu algoritma greedy merupakan metode yang paling populer untuk memecahkan persoalan optimasi [4]. Ada dua persoalan optimasi, yaitu optimasi minimum dan maksimum, yang digunakan pada hal ini adalah optimasi minimum. Algoritma greedy pada pembentukan pohon di sini yaitu membagi dua pohon menjadi frekuensi yang lebih kecil, kemudian hubungkan pada sebuah akar. Akar tersebut kemudian dipisah kembali dan digabung dengan akar yang berada di atasnya (akar baru).
3. proses ketiga yaitu proses rekursif dari proses kedua sehingga akar utama pohon memiliki frenuensi bernilai 1.

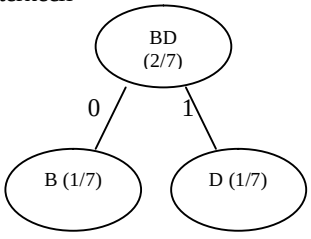
Inti dari algortima huffman adalah menggunakan priority queue yang direkursif sehingga membentuk pohon dengan kompleksitas waktu On(log n)[3].

Proses-proses ini akan lebih jelas dengan contoh gambar.

Misal karakternya adalah ABACCDA. Jika kita mengikuti kode ascii tanpa menggunakan algoritma huffman, maka kita akan memakan banyak memori yaitu 7bit x 8 karakter = 56 bit. Tapi dengan pohon huffman, kita akan lihat bedanya. Yang pertama dilakukan dalam menbentuk pohon huffman yaitu hitung frekuensi dari tiap-tiap karakter

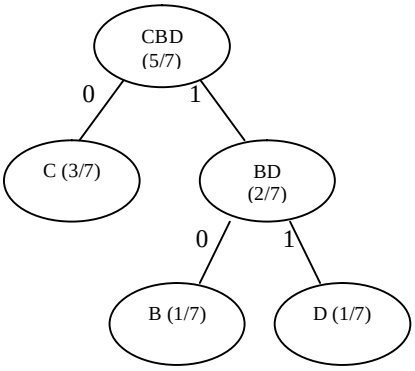
A : 3/7
B : 1/7
C : 2/7
D : 1/7

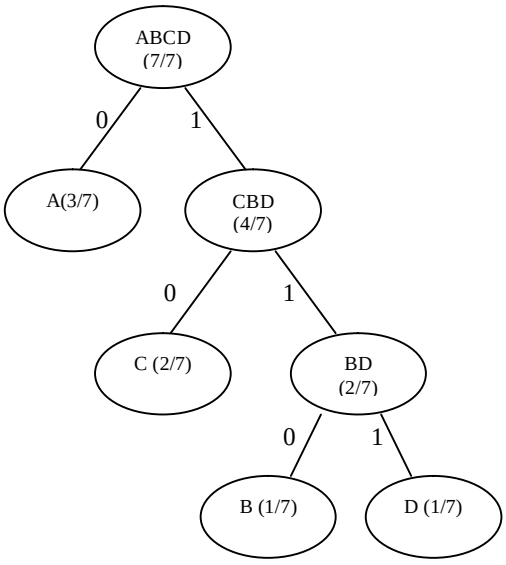
Kemudian lakukan proses kedua dari proses-proses di atas yaitu buatlah simpul dan akar dari frekuensi yang terkecil



Kemudian proses ini diulang (rekursif) hingga semua karakter terbentuk menjadi pohon.

Dapat dilihat pada contoh pembentukan phon di bawah ini:





Pada akhirnya pohon jadi dan tiap-tiap karakter telah memiliki identitas yang akan digunakan untuk proses encoding (proses selanjutnya).

2.1.2 Proses Encoding

Setelah melihat hasil dari pohon, dapat kita membuat sebuah tabel yang berisi identitas dari tiap-tiap karakter berdasarkan biner. Dalam proses encoding ini, prosesnya adalah baca urutan biner tiap karakter hingga sampai edge dari pohon yang berisi karakter yang dicari. Misal, akan mencari huruf B, maka baca

urutan karakter dari akar hingga huruf B yaitu 110. begitu pula untuk seluruh karakter-karakternya. Sehingga dapat dibentuk tabel sebagai berikut :

Karakter	Binary
A	0
B	110
C	10
D	111

Jika kita cermati, hasil dari dari peng-encodingan ini adalah membuat sebuah rangkaian karakter yang sama dengan bit yang lebih minim. Dari ABACCDA yang seharusnya 56 bit, sekarang menjadi 13 bit saja. Itulah guna dari algoritma Huffman.

2.1.3 Proses Decoding

Proses decoding ialah kebalikan dari encoding. Dalam sistem kerjanya tidak ada perbedaan yang signifikna dengan sistem kerja encoding. Hanya kalau encoding membaca pohon dari bawah (edge) hingga akar, kalau decoding dari akar hingga ketemu edge atau dalam hal ini karakter. Analoginya adalah seperti ini, decoding adalah jalur yang dilewati untuk menemukan sebuah benda. Misal kita diberikan jalur 111, maka kita urutkan jalur tersebut pada pohon sehingga menemukan benda (dalam hal ini karakter). Seperti itulah inti kerja dari decoding.

2.2 Cara Kerja Algoritma Shannon-Fano

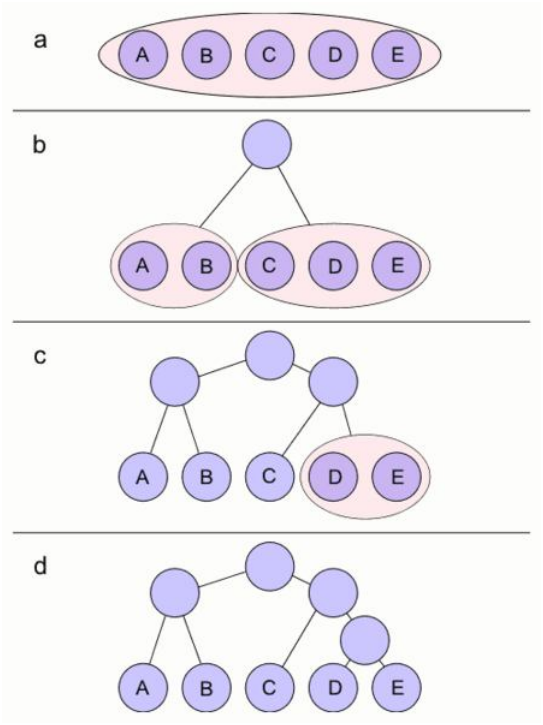
Pada dasarnya cara kerja dari algoritma Shannon-Fano ini sama persis dengan Huffman. Algoritma ini membentuk sebuah pohon, kemudian meng-encodingnya, dan yang terahir adalah megnembalikannya dalam bentuk karakte teks atau decoding. Hanya saja perbedaan yang fundamental terdapat pada pembuatan pohon. Pembuatan pohon pada Shannon-Fano dibuat berdasarkan proses dari atas ke bawah – berbeda dengan huffman yang membuat pohon dari bawah ke atas. Sebuah pohon Shannon-Fano dibangun sesuai dengan spesifikasi yang dirancang untuk mendefinisikan tabel kode yang efektif. Algoritma yang sebenarnya sederhana:

1. Untuk daftar simbol-simbol tertentu, mengembangkan sebuah daftar yang sesuai probabilitas atau frekuensi dihitung sehingga setiap simbol frekuensi relatif kejadian diketahui.
2. Menyortir daftar simbol-simbol sesuai dengan frekuensi, dengan simbol-simbol yang paling sering terjadi di sebelah kiri dan yang paling umum di sebelah kanan.
3. Membagi daftar menjadi dua bagian, dengan total frekuensi dihitung dari kiri setengah menjadi seperti dekat dengan jumlah yang tepat mungkin.
4. Kiri setengah dari daftar ditetapkan angka biner 0, dan hak diberikan setengah digit 1. Ini berarti bahwa kode untuk simbol-simbol pada semester pertama semua akan dimulai dengan 0, dan aturan-aturan di paruh kedua akan semua mulai dengan 1.
5. Rekursif menerapkan langkah 3 dan 4 untuk masing-masing dari dua bagian, membagi kelompok dan menambahkan kode bit sampai setiap simbol telah menjadi kode yang sesuai daun di pohon.

Untuk mempermudah, diilustrasikan dengan gambar berikut.

Contoh menunjukkan pembangunan kode Shannon alfabet kecil. Kelima simbol-simbol yang dapat dikodekan memiliki frekuensi berikut: A = 15 ; B = 7 ; C = 6 ; D = 6 ; E = 5

Semua simbol-simbol yang diurutkan berdasarkan frekuensi, dari kiri ke kanan (ditunjukkan pada Gambar a). Menempatkan garis pemisah antara simbol-simbol B dan C menghasilkan total 22 di grup kiri dan total 17 di kelompok yang tepat. Ini meminimalkan perbedaan total antara dua kelompok. Dengan pembagian ini, A dan B akan masing-masing memiliki kode yang dimulai dengan 0 bit, dan C, D, dan E kode akan semua mulai dengan 1, seperti ditunjukkan pada Gambar b. Kemudian, di sebelah kiri setengah dari pohon mendapat divisi baru antara A dan B, yang menempatkan A pada daun dengan kode 00 dan B pada daun dengan kode 01. Setelah empat divisi prosedur, pohon hasil kode. Pohon di final, tiga simbol dengan frekuensi tertinggi semuanya telah ditugaskan 2-bit kode, dan dua simbol dengan tuntutan yang lebih rendah memiliki 3-bit kode seperti ditunjukkan tabel di bawah ini:



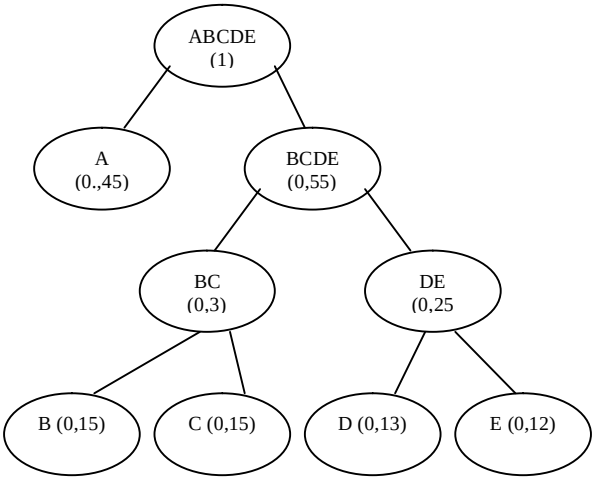
Setelah pohon jadi, maka proses selanjutnya adalah sama yaitu encoding dan decoding.

2.3 Kelebihan dan Kekurangan

Dilihat dari cara dalam pembentukan pohon, sudah tampak jelas perbedaan antara kedua algoritma tersebut. Algoritma huffman memebentuk pohon dari bawah sampai atas atau dengan kata lain dari simpul hingga akar, sedang algortima shannon-fano membentuk pohon dari akar ke simpul. Dalam pembentukan prioritas dari pohon yang dibenuk juga berbeda. Pada kasus sebelumnya, pohon huffman terbantuk dengan mengelompokkan karakter string yang memiliki frekuensi rendah satu per satu kemudian membentuknya menjadi simpul dan merekursifnya. Namun, pada pohon shannon-fano, semua karakter dikelompokkan berurutan dari kiri ke kanan dari frekuensi yang sering muncul ke frekuensi yang umum. Untuk mempermudah dalam melihatperbedaan efisiensi dalam kedua algoritma, kita dapat lihat dari contoh kasus yang direpresentasikan pada pohon berikut ini.

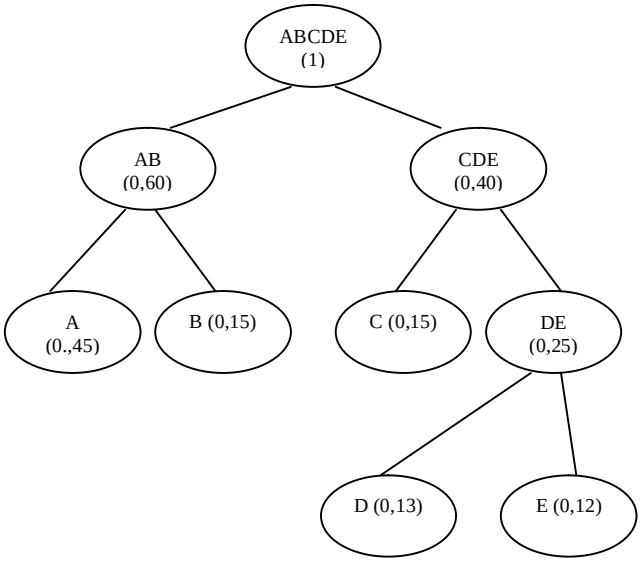
Misal terdapat lima karakter A; B; C; D; E.
Frek A : 0,45
Frek B : 0,15
Frek C : 0,15
Frek D : 0,13
Frek E : 0,12.

Jika direpresentasikan pada pohon huffman maka akan menjadi sebagai berikut :



Melihat dari hasil pembentukan bit baru (hasil encoding), karakter A yang sering muncul (probabilitas hingga 0,45) hanya memiliki satu bit saja yaitu 0. Kompresi seperti ini yang membuat algoritma menjadi lebih efektif karena pohon terbentuk dari bawah (probabilitas kecil atau umum) ke atas (probabilitas besar atau sering).

Sedangkan pada pohon shannon-fano akan terbentuk seperti ini :



Secara sekilas, pohon shannon-fano ini memang tampak lebih ringkas, tetapi jika kita amati baik-baik, terdapat kelemahan di dalamnya.
Untuk memperjelas, penulis akan mengilustrasikan dengan angka, misal total dari seluruh kemunculan karakter tersebut adalah 100, maka

Karakter	A	B	C	D	E
Kemunculan	45	15	15	13	12
Total					100

Berdasarkan pohon huffman, maka encoding menjadi

Karakter	A	B	C	D	E
Huffman code	0	100	101	110	111

Penghitungan kapasitas memori yang dipakai yaitu :
Kemunculan karakter X jumlah bit
Total memori (untuk ABCDE) =
45x1bit + 15x3bit + 15x3bit + 13x3bit + 12x3bit =
210 bit

Berdasarkan pohon shannon-fano, maka proses encoding menjadi

Karakter	A	B	C	D	E
Shannon-fano code	00	01	10	110	111

Dengan perhitungan jumlah kapastias memori yang sama seperti di atas, maka didapat untk total memori yaitu
45x2bit + 15x2bit + 15x2bit + 13x3bit + 12x3bit =
225 bit.

Dari perhitungan kapasitas memori yang digunakan setelah proses decoding dilakukan, terlihat beda yang mungkin tidak terlihat besar. Namun, ketika itu adalah sebuah dokumen yang berisi ratusan halaman atau berupa file gambar yang berisi jutaan pixel, maka angka dari kapasitas memori ini sangat berperan dalam menentukan kompresi yang lebih efektif.

Ini lebih disebabkan karena algoritma huffman dapat membentuk bentuk prefix yang lebih efisien dibandingkan dengan shannon-fano[2]. Kalau dibandingkan dengan kode ASCII berarti algoritma huffman hanya menggunakan 30% bit saja. Karena satu karakter memiliki 7 bit pada kode ASCII, berarti pada kemunculan 100 kali terdapat 700 bit. Huffman 210 bit sedang shannon-fano 235 bit. Dengan kata lain huffman dapat mengompres file hingga 70% dan Shannon-fano hanya dapat mengopres file sebesar 66,5% berdasarkan pengujian di atas. Hasil ini memang tidak dapat dipukul rata untuk segala file, tapi ini sebagai gambaran untuk pembanding saja. Algoritma shannon-fano dapat bekerja sama efektifnya dengan algoritma huffman pada kasus tertentu (tergantung dari probabilitas kumunculan karakter).

Tidak berarti juga algoritma shannon-fano tidak digunakan dalam kompresi komputasi, algoritma ini ternyata digunakan pada proses pembentukan ZIP file[2].

3. KESIMPULAN

Pada dasarnya, semua perangkat kompresi adalah bekerja dengan proses yang sama. Hanya saja terdapat yang lebih efektif dibandingkan yang lain. Algoritma huffman memiliki kelebihan dalam pembuatan pohon yang efektif sehingga kompresi teks dapat menjadi lebih minimum (bekerja berdasarkan prinsip kerja algoritma greedy). Pada pembentukan pohon huffman, pohon dibuat dari simpul hingga ke akar, sedang pada pohon shannon-fano terbentuk dari akar ke simpul. Ini yang menyebabkan kompresi menjadi berbeda untuk karakter yang sama. Tingkat efisiensi pada algoritma huffman mencapai 70%, sedangkan tingkat efisiensi algoritma shannon-fano hanya 66,5% (berdasarkan uji coba yang dilakukan oleh penulis). Kompleksitas waktu dari algoritma huffman adalah O n(log n), sedang kompleksitas dari shannon-fano tidak diketahui karena tidak ada pada referensi manapun.

Kembali pada sejarahnya, algoritma shannon-fano ini ditemukan setelah Huffman (yang merupakan murid dari Shannon) menemukan terlebih dahulu algoritma kompresinya. Prof Shannon ingin membuat algoritma tandingan dari algoritma yang dibuat oleh mahasisanya, tetapi pada kenyataannya masih terdapat kekurangannya.

Contoh aplikasi yang menggunakan algoritma kompresi ini (baik shannon-fano dan huffman) yaitu pada kompresi komputasi (zip file), kompresi MP3, kompresi gambar dengan pixel,dan lain sebagainya.

DAFTAR REFERENSI

[1] Munir, Rinaldi, “Diktat Kuliah IF2151 Matematika Diskrit edisi IV”, Departemen Teknik Informatika Institut Teknologi Bandung, 2004, hal.IX-26 – IX-28.

[2]http://en.wikipedia.org/wiki/Shannon%E2%80%93Fano_coding tanggal akses : 16 Desember 2009 pukul 21.20

[3]http://en.wikipedia.org/wiki/Huffman_coding tanggal akses : 16 Desember 2009 pukul 21.20

[4]http://74.125.153.132/search?q=cache:4mjOMcdESvYJ:mufidnilmada.staff.gunadarma.ac.id/Downloads/files/9711/Algoritma%2BGreedy.ppt+algoritma+greedy&cd=4&hl=id&ct=clnk&gl=id&client=firefox-a Tanggal akses : 17 Desember 2009 pukul 21.00

[5] http://docs.google.com/viewer?a=v&q=cache:gBh_mgjTwZEJ:journal.uii.ac.id/index.php/media-informatika/article/view/115/77+waktu+shannon-fano&hl=id&gl=id&pid=bl&srcid=ADGEESi7Y_rabDmw2ywmi2yIiKbyOGZsxRpQeffvgJU0MkWb-WUzggYV7Wp6fIZwsZBVgsjiPLORZUmqlffozqg8JBlEm8J44cCB-LhqgtP3SoZyCYk1b_Skhl3lgMwmmZC2BuAWQX9i&sig=AHIEtbT7oHuB2ju4APIrTH6rOn9OoUmVtQ tanggal akses 17 desember 2009 pukul 21.05