PEMAMPATAN DATA DENGAN KODE HUFFMAN (APLIKASI POHON BINER)

Winda Winanti (13505017)

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if115017@students.if.itb.ac.id

Abstraksi

Dalam komunikasi data, pesan yang dikirim seringkali ukurannya sangat besar sehingga waktu pengirimannya lama. Begitu juga dengan penyimpanan data, arsip yang berukuran besar membutuhkan ruang penyimpanan yang besar. Kedua masalah ini dapat diatasi dengan mengkodekan pesan atau isi arsip sesingkat mungkin, sehingga waktu pengiriman pesan relatif cepat dan ruang penyimpanan yang dibutuhkan juga sedikit. Cara pengkodean seperti ini disebut kompresi (pemampatan) data. Salah satu cara kompresi data ini adalah dengan menggunakan salah satu aplikasi pohon biner yaitu kode Huffman.

Kata kunci :
Huffman, pemampatan, pohon biner.

1. Pendahuluan

Dalam kehidupan sehari-hari, terutama dalam bidang teknologi informasi, perpindahan data, pesan, dan informasi sangat sering dilakukan. Perpindahan itu dapat dilakukan melalui media penyimpanan (seperti *floppy disk*, *hard disk*, CD-ROM, *flash disk*) ataupun melalui media internet. Terkadang kendala dalam perpindahan data tersebut adalah besarnya ukuran *file* yang akan dipindahkan sangat besar, sehingga ada resiko tidak dapat tertampung pada media penyimpanan dan tidak tersampaikan nya data dalam proses *transfer* data. Hal ini tentu saja tidak kita inginkan.

Untuk menanggulangi kedua hal tersebut, biasanya pengguna komputer memakai aplikasi pemecah-mecah data (*file* splitter) namun hal ini tidak efektif dan efisien karena ukuran *file* tidak berubah, namun hanya dipotong-potong menjadi beberapa bagian. Cara lain yang lebih umum dan baik untuk digunakan untuk menanggulangi masalah ini adalah dengan menggunakan aplikasi pemampat data untuk memampatkan data sehingga ukuran *file* dapat dijadikan lebih kecil.

Beberapa aplikasi pemampat data yang terkenal dan biasa digunakan oleh para pengguna komputer adalah WinZip (menghasilkan format .zip) dan WinRAR (menghasilkan format .rar) atau bahkan menggunakan pemampat data yang sudah tersedia pada *console* di sistem operasi LINUX (mengasilkan format .tar.gz). Ketiga contoh aplikasi pemampat data di atas bisa digunakan untuk memampatkan data dengan baik dan sudah dapat dipercaya.

Aplikasi pemampat data menggunakan cara pemampatan (kompresi) dan penirmampatan (dekompresi). Kompresi data atau pemampatan data adalah proses transformasi dari *string* ke *string* yang memiliki informasi sama namun memiliki panjang yang lebih sedikit (pendek). Salah satu cara kompresi data ini adalah dengan menggunakan salah satu aplikasi dari pohon biner (*binary tree*) yaitu kode Huffman.

Dalam makalah ini, penulis ingin menjelaskan bagaimana kode Huffman dapat memampatkan data dan bag. Namun sebelum penulis menjelaskan tentang kode Huffman, penulis akan menjelaskan dengan singkat teori dasar pohon biner dengan harapan pembaca lebih memahami isi penulisan tentang pemampatan data dengan kode Huffman.

2. Pohon Biner (*Binary Tree*)

Definisi Pohon
Definisi formal dari pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit. Struktur pohon adalah struktur data yang penting di bidang informatika dan pemrograman. Struktur pohon memungkinkan kita untuk mengorganisasi informasi berdasarkan suatu struktur lojik dan memungkinkan bebagai cara akses untuk suatu elemen.

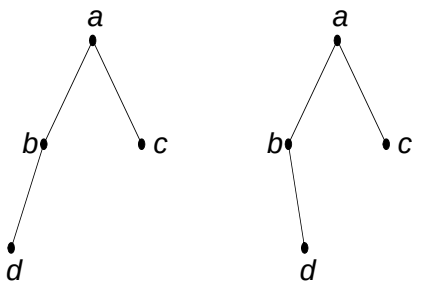
Sifat Pohon
Misalkan $G = (V, E)$ adalah graf tak-berarah sederhana dan jumlah simpulnya n. Maka, semua pernyataan di bawah ini adalah ekivalen:

1. G adalah 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 memiliki $m = n - 1$ buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan.

Beberapa istilah penting dalam pohon adalah :

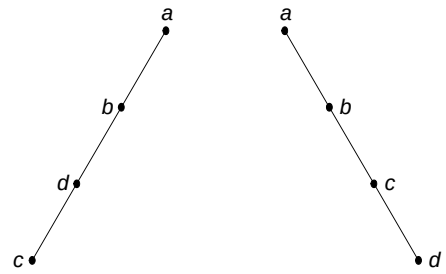
- Hutan : kumpulan pohon yang saling lepas
- Anak (child) : subpohon dari sebuah akar
- Orangtua (parent) : akar dari sebuah subpohon
- Upapohon (subtree) : upagraf sebuah pohon sedemikian hingga upagraf tersebut mengandung x dan semua keturunannya
- Derajat (degree) : jumlah upapohon (jumlah anak) pada suatu simpul
- Ketinggian (level) : panjangnya jalan dari akar sampai dengan simpul yang bersangkutan
- Daun (leaf) : simpul terminal dari pohon
- Simpul (node) : elemen dari pohon yang memungkinkan akses pada subpohon dimana simpul tersebut berfungsi sebagai akar
- Kedalaman (depth) : panjang maksimum jalan dari akar menuju ke sebuah daun

Pohon Biner
Pohon biner adalah pohon yang setiap simpulnya memiliki paling banyak dua buah anak yaitu kiri (*left*) dan kanan (*right*). Alih-alih menyebutnya anak pertma dan anak kedua dari suatu simpul dalam, dapat disebut anak kiri (*left child*) dan anak kanan (*right child*). Pohon yang akarnya adalah anak kiri disebut upapohon kiri (*left subtree*), sedangkan pohon yang akarnya adalah anak kanan disebut upapohon kanan (*right subtree*). Karena adanya perbedaan anak/upapohon kiri dan anak/upapohon kanan, maka pohon biner adalah pohon terurut.



Gambar Dua buah pohon biner yang berbeda

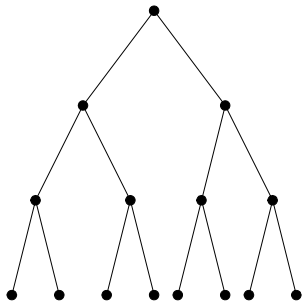
Pohon yang semua simpulnya terletak dibagian kiri saja atau dibagian kanan saja disebut pohon condong (*skewed tree*). Pohon yang condong ke kiri disebut pohon condong-kiri (*skew left*), sedangkan pohon yang condong ke kanan disebut pohon condong kanan (*skew left*).



Gambar (a) Pohon condong-kiri (b) Pohon condong-kanan

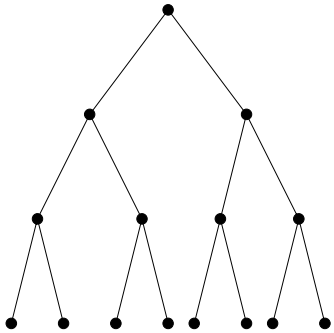
Dua jenis pohon biner :

- Pohon biner penuh
  Merupakan pohon biner yang setiap simpulnya tepat memiliki dua buah anak yaitu kiri dan kanan kecuali pada daun (simpul terbawah).

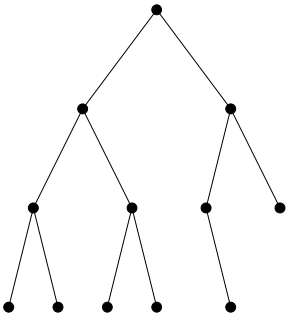


Gambar Pohon biner penuh

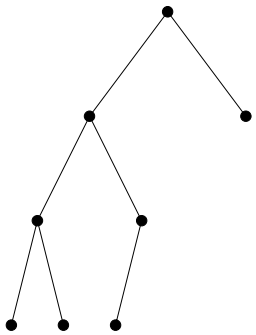
- Pohon biner seimbang
  Pohon yang memiliki perbedaan tinggi antara upapohon kanan dan upapohon kiri maksimal 1.



Gambar Pohon biner seimbang



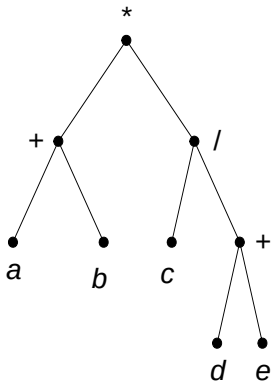
Gambar Pohon biner seimbang



Gambar Bukan pohon biner seimbang

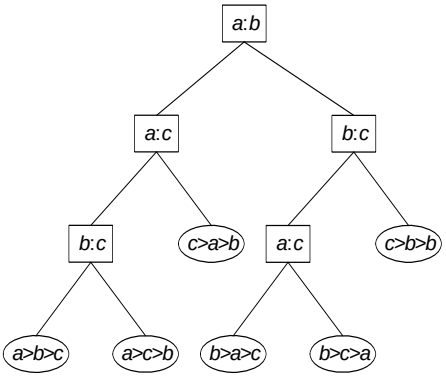
Terapan Pohon Biner

1. Pohon Ekspresi
   Pohon ekspresi ialah pohon biner dengan daun berupa *operand* dan simpul dalam (termasuk akar) berupa operator.



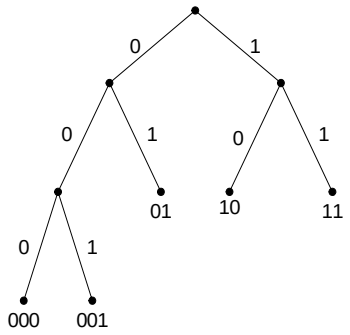
Gambar Pohon ekspresi dari *(a + b)*(c/(d + e))*

2. Pohon Keputusan
   Pohon keputusan digunakan untuk memodelkan persoalan yang terdiri dari serangkaian keputusan yang mengarah ke solusi. Tiap simpul dalam menyatakan keputusan, sedangkan daun menyatakan solusi.



Gambar Pohon keputusan untuk mengurutkan 3 buah elemen

3. Kode Awalan
   Kode awalan (*prefix code*) adalah himpunan kode, misalnya kode biner, sedemikian sehingga tidak ada anggota kumpulan yang merupakan awalan dari anggota yang lain.



Gambar Pohon biner dari kode prefiks { 000, 001, 01, 10, 11}

4. Kode Huffman
   Kode Huffman suatu teknik yang dapat digunakan untuk mengkonstruksi kode prefiks.

Tabel Kode ASCII

Simbol	Kode ASCII
A	01000001
B	01000010
C	01000011
D	01000100

rangkaian bit untuk string '*ABACCDA*':

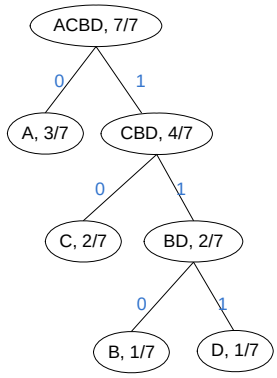
01000001010000010010000010100000110100000110100010001000001

Tabel Tabel kekerapan dan kode Huffman untuk *string 'ABACCDA'*

Simbol	Kekerapan	Peluang	Kode Huffman
A	3	3/7	0
B	1	1/7	110
C	2	2/7	10
D	1	1/7	111

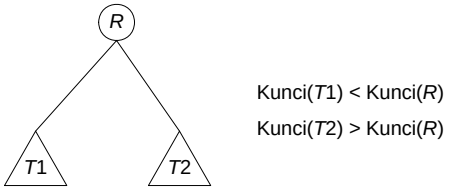
rangkaian bit untuk *'ABACCDA'*:

0110010101110

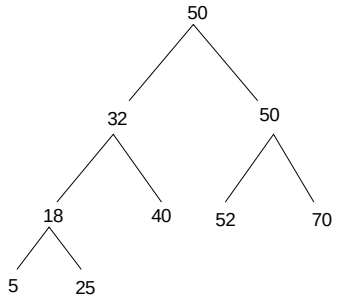


Gambar Pohon Huffman untuk pesan 'ABACCDA'

5. Pohon Pencarian Biner
   Pohon pencarian biner (*binary search tree* – BST) mungkin adalah pohon biner yang paling penting, khususnya pada persoalan yang banyak melakukan operasi pencarian, penyisipan, dan penghapusan elemen.



Gambar Skema pohon pencarian



Gambar Contoh pohon pencarian

3. Pemampatan Data dengan Algoritma Huffman

Keterkaitan Pemampatan Data dan Pohon Huffman

Pemampatan data meliputi transformasi string ke string lain yang memiliki informasi yang sama tetapi dengan panjang yang lebih sedikit. Ini membutuhkan perancangan dari kode yang dapat digunakan untuk secara unik merepresentasikan setiap karakter pada string masukan. Lebih tepat, kode dikatakan

memetakan pesan sumber ke kata yang telah dikodekan. Proses untuk menggunakan kode untuk mentransformasikan pesan semua ke kata yang telah dikodekan dinamakan *encoding*, sementara mentransformasi-balik dari kata yang telah dikodekan menjadi pesan semula dinamakan *decoding*. Perancangan skema pengkodean yang efisien adalah sangat penting sebab dapat secara signifikan mengurangi jumlah memori yang digunakan untuk menyimpan berkas data serta jumlah waktu yang dibutuhkan untuk menyimpan berkas data serta jumlah waktu yang dibutuhkan untuk mentransmisikan data tersebut.

Jika *fixed-length code* digunakan, maka kode yang dihasilkan akan memiliki panjang yang sama. Kode ASCII adalah kode dengan panjang tetap yang memetakan 256 karakter-karakter yang berbeda ke kode 8 bit. Perhatikan jika p adalah himpunan karakter yang mungkin pada masukan, maka setiap kode pada pengkodean panjang tetap membutuhkan paling tidak $\lceil \log p \rceil$ bit untuk mengkodekan setiap karakter.

Tabel Kode ASCII

Simbol	Kode ASCII
A	01000001
B	01000010
C	01000011
D	01000100
E	01000101
F	01000110
G	01000111
H	01001000
I	01001001
J	01001010
K	01001011
L	01001100
M	01001101
N	01001110
O	01001111
P	01010000
Q	01010001
R	01010010
S	01010011
T	01010100
U	01010101
V	01010110
W	01010111
X	01011000
Y	01011001
Z	01011010

Sebagai contoh, untuk mendapatkan string ‘WINDA<spasi>WINANTI’ :
(Kode ASCII untuk spasi adalah 00100000)

01010111 01001001 01001110 01000100 01000001 0010000 01010111 01001001 01001110 01000001 01001110 01010100 01001001

Adalah mungkin untuk secara signifikan mengurangi jumlah yang dibutuhkan untuk merepresentasikan pesan sumber jika *variable length code* digunakan. Pada kasus ini, jumlah bit yang dibutuhkan bervariasi untuk masing-masing karakter. Sasarannya adalah mengkodekan karakter-karakter yang muncul lebih sering menggunakan string bit yang lebih pendek dan karakter yang muncul agak jarang dengan jumlah bit yang lebih panjang.

Suatu metoda pemampatan data mencakup perhitungan frekuensi dari semua karakter pada pesan yang diberikan sebelum pesan tersebut dikirim atau disimpan. Kemudian, kode *prefix* dengan panjang bervariasi akan dikonstruksi dengan karakter-karakter yang paling sering muncul dikodekan secara lebih pendek dibandingkan karakter-karakter yang jarang muncul.

Adalah berguna untuk berpikir bahwa pemampatan data sebagai masalah optimasi jumlah dari bit yang digunakan untuk mengkodekan pesan. Jika kita mengasumsikan M sebagai pesan yang mengandung karakter-karakter dari sekumpulan alfabet Γ dan pohon biner T berhubungan dengan kode *prefix* dari alfabet, maka kita anggap $f_M(c)$ mencatat banyak kemunculan (frekuensi) karakter c dalam M dan $d_T(c)$ kedalaman daun yang menyimpan karakter c dalam T, maka fungsi ‘biaya’ (jumlah bit yang dibutuhkan) untuk mengkodekan M menggunakan T adalah:

$$C_M(T) = \sum_{c \in \tau} f_M(c) d_T(c)$$

Minimasi fungsi biaya ini akan menghasilkan kode *prefix* optimal untuk M. Kita akan merujuk pada pohon biner yang merrepresentasikan kode *prefix* optimal ini sebagai pohon optimal.

Kita sekarang akan melakukan analisis algoritma *greedy* untuk menggunakan antrian berprioritas untuk menciptakan kode *prefix* optimal untuk pesan. Pohon optimal yang

dihasilkan dinamakan Pohon Huffman dan kode *prefix* yang berhubungan dengan pohon itu kita namakan sebagai Kode Huffman. Perhatikan bahwa keunggulan pendekatan dengan algoritma *greedy* adalah untuk menghasilkan algoritma yang efisien. Tidak sulit untuk memahami bahwa ini adalah faktor yang penting saat mentransmisikan data. Jika pemampatan data berlangsung terlalu lama, kita tidak akan mendapatkan keunggulan apapun dari pentransmisian data.

Prosedur Huffman

Pada algoritma dibawah ini, kita mengasumsikan bahwa untuk setiap karakter $c \epsilon \Gamma$, frekuensi $f_M(c)$ sudah dihutung. Pembaca sesunggunya dapat melihat bahwa frekuensi-frekuensi itu dapat dihitung dan bersifat linier terhadap ukuran pesan $|M|$. Kemudian, untuk setiap $c \epsilon \Gamma$ dengan nilai $f_M(c)$ tidak sama dengan 0, pohon biner diciptakan dan *field* data diinisialisasi dengan $f_M(c)$. Lalu dirujuk himpunan simpul-simpul sebagai Γ_M. Field data untuk setiap simpul $v \epsilon \Gamma_M$ dapat diakses dengan $f_M(v)$. Koleksi dari simpul-simpul membentuk hutan dari pohon biner bersimpul tunggal. Kemudian, ide dasarnya adalah membentuk pohon biner bersimpul $|\ \Gamma_M\ |$ dengan secara berulang membentuk pohon lebih besar dari 2 pohon-pohon dalam hutan dengan prioritas yang lebih tinggi. Prioritas pohon dihitung dari jumlah frekuensi-frekuensi karakter-karakter yang disimpan pada daun-daunnya. Ini adalah strategi *greedy* karena setiap langkahnya menghasilkan fungsi biaya paling rendah.

```
Huffman (vertex set Γ_M)
1 ANTRIAN BERPRIORITAS P ← Γ_M
2 for i← 1 to |Γ_M|-1 do
3   v ← ciptakan simpul pohon biner
    baru
4   left[v] ← DeleteMin(P)
5   right[v] ← DeleteMin(P)
6   f_M[v] ← f_M [left[v]] + f_M
    [right[v]]
7   Insert(P,m)
8 endfor
9 return DeleteMin(P)
```

Baris pertama prosedur `Huffman()` menciptakan antrian berprioritas dan menginisialisasinya dengan hutan dari pohon biner bersimpul tunggal sejumlah Γ_M. Kalang 2 hingga 8 secara berulang menghapus 2 pohon dari antrian berprioritas yang memiliki prioritas lebih rendah (baca : frekuensi lebih rendah). Dua pohon ini menjadi subpohon kiri dan kanan dari pohon biner dengan simpul akan `v`. Prioritas dari pohon ini, yang tersimpan dalam *field* data yang dimiliki oleh `v`, kemudian diatur sebagai jumlah dari prioritas subpohon. Pohon baru ini kemudian disisipkan pada antrian berprioritas. Adalah mudah untuk melihat sejumlah $|\Gamma_M|-1$ perulangan, semua elemen pada antrian berprioritas akan menjadi pohon biner bersimpul $|\Gamma_M|$.

Contoh Pemampatan Data dengan Pohon Huffman

Misalkan string masukan adalah ‘WINDA WINANTI’

Dari tabel diatas, untuk mendapatkan string ‘WINDA<spasi>WINANTI’ :
(Kode ASCII untuk spasi adalah 00100000)

01010111 01001001 01001110 01000100 01000001 0010000 01010111 01001001 01001110
01000001 01001110 01010100 01001001

atau jika dihitung besar ukuran arsip nya adalah 13 × 8 = 104 bit (13 *byte*).

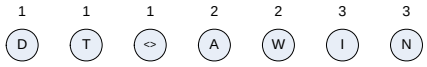
Jumlah itu dapat dikompresi lagi dengan menggunakan kode Huffman yaitu :

- Buat tabel kekerapan dari string WINDA<spasi>WINANTI

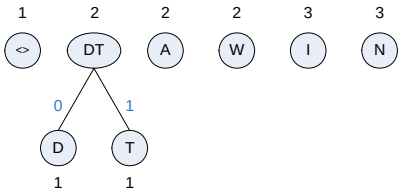
Simbol	Kekerapan	Peluang
A	2	2 / 13
D	1	1 / 13
I	3	3 / 13
N	3	3 / 13
T	1	1 / 13
W	2	2 / 13
<spasi>	1	1 / 13

Langkah selengkapnya disajikan dalam gambar di bawah ini :

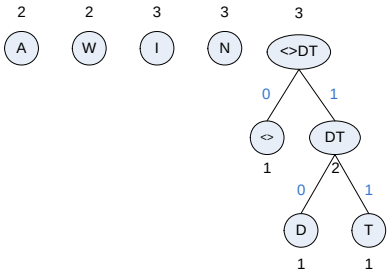
1. Buat semua karakter menjadi simpul bebas dan sertakan kekerapan muncul karakter tersebut dalam string. Dalam contoh ini string yang digunakan adalah ‘WINDA<>WINANTI’. Maka simpulnya setelah diurut dari terkecil hingga terbesar menurut kekerapannya adalah seperti gambar di bawah ini :



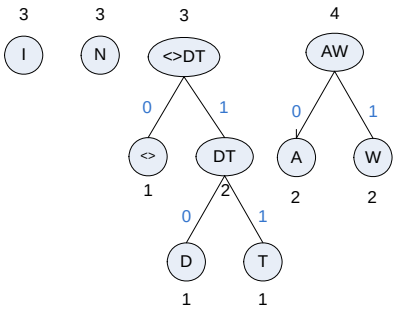
2. Pilih dua simpul dengan kekerapan paling kecil lalu gabungkan kedua simpulnya menjadi simpul baru. Dalam contoh ini dua simpul terkecil adalah D dan T, lalu gabungkan menjadi simpul baru dengan kekerapan 1 + 1 = 2. Setelah simpul DT terbentuk maka urutkan lagi simpul DT dengan simpul sebelumnya sehingga menjadi seperti :



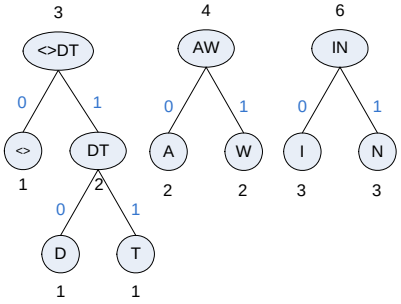
3. Lakukan kembali langkah 2 untuk semua simpul hingga terbentuk pohon Huffman. Dengan demikian maka langkah berikutnya adalah menggabungkan dua simpul dengan kekerapan terkecil yaitu simpul <spasi> dan DT sehingga menghasilkan simpul baru yaitu <spasi>DT dengan kekerapan 1 + 2 = 3. Setelah simpul <spasi>DT terbentuk maka urutkan lagi simpul <spasi>DT dengan simpul sebelumnya sehingga menjadi seperti :



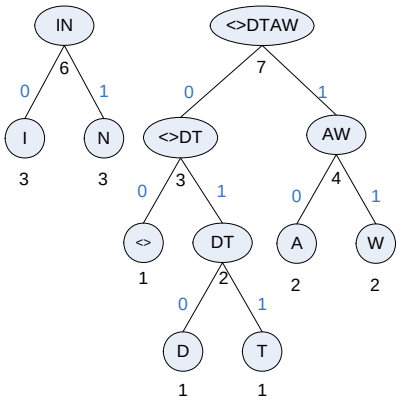
4. Gabungkan dua simpul dengan kekerapan terkecil yaitu simpul A dan W sehingga menghasilkan simpul baru yaitu AW dengan kekerapan 2 + 2 = 4. Setelah simpul AW terbentuk maka urutkan lagi simpul AW dengan simpul sebelumnya sehingga menjadi seperti :



5. Gabungkan dua simpul dengan kekerapan terkecil yaitu simpul I dan N sehingga menghasilkan simpul baru yaitu IN dengan kekerapan 3 + 3 = 6. Setelah simpul IN terbentuk maka urutkan lagi simpul IN dengan simpul sebelumnya sehingga menjadi seperti :

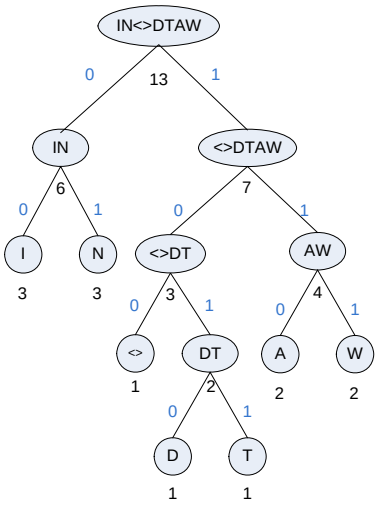


6. Gabungkan dua simpul dengan kekerapan terkecil yaitu simpul <spasi>DT dan AW sehingga menghasilkan simpul baru yaitu <spasi>DTAW dengan kekerapan 3 + 4 = 7. Setelah simpul <spasi>DTAW terbentuk maka urutkan lagi simpul <spasi>DTAW dengan simpul sebelumnya sehingga menjadi seperti :



7. Gabungkan dua simpul dengan kekerapan terkecil yaitu simpul IN dan <spasi>DTAW sehingga menghasilkan simpul baru yaitu IN<spasi>DTAW

dengan kekerapan 6 + 7 = 13. Simpul ini sudah menjadi akar dari pohon Huffman dan pembentukan pohon telah selesai.



Simbol	Keke-rapan	Peluang	Kode Huffman
A	2	2 / 13	110
D	1	1 / 13	1010
I	3	3 / 13	00
N	3	3 / 13	01
T	1	1 / 13	1011
W	2	2 / 13	111
<spasi>	1	1 / 13	100

Sehingga string ‘WINDA WINANTI’ direpresentasikan menjadi
111 00 01 1010 110 100 111 00 01 110 01 1011 00

Dan dapat diketahui ukuran arsip menjadi 3+2+2+4+3+3+3+2+2+3+2+4+2 = 35 bit (4,375 byte).

Analisis Prosedur Huffman

Untuk analisis prosedur `Huffman`, kita asumsikan bahwa antrian berprioritas P diimplementasikan menggunakan *heap*. Mari kita asumsikan $|\Gamma| = n$. Kita sudah melihat bahwa himpunan Γ_M dapat dikonstruksi dalam $O(n)$ satuan waktu dan pada kasus terburuk $|\Gamma_M| = n$. Pada kasus ini, kalang 2 hingga 8 yang terdapat pada prosedur `Huffman` akan dieksekusi dalam $n-1$ satuan waktu. Pada setiap iterasi, 3 antrian berprioritas dibentuk, masing-masing membutuhkan $O(\log n)$ satuan waktu. Sehingga kalang mengkontribusi $O(n \log n)$ ke waktu eksekusi total. Sehingga, keseluruhan waktu eksekusi dari prosedur `Huffman()` adalah $O(n \log n)$ satuan waktu.

Untuk memperlihatkan bahwa prosedur Huffman diatas menghasilkan pohon yang optimal, kita harusmemeprlihatkan bahwa urutan dari strategi *greedy* akan meminimisasi fungsi biaya C_M seperti yang diperlihatkan pada persamaan diatas. Kita akan mengerjakannya menggunakan induksi dari n, dimana n adalah ukuran dari himpunan simpul Γ_M. Dalam hal ini, adalah mudah untuk memverifikasi bahwa prosedur `Huffman` diatas menghasilkan pohon optimal saat n sama dengan 1 atau 2. Hipotesis induksi yang diambil adalah bahwa prosedur `Huffman` tersebutakan menghasilkan pohon optimal untuk setiap keadaan dimana $|\Gamma_M|$ tidak lebih besar dari $n - 1$. Langkah perluasannya adalah mencakup langkah untuk menunjukkan bahwa jika prosedur `Huffman` diatas menghasilkan pohon optimal saat $|\Gamma_M| = n - 1$, maka prosedur `Huffman` tersebut juga akan menghasilkan pohon optimal saat $|\Gamma_M| = n$. Kita akan memperlihatkan langkah ini adalah persoalan yang ekivalen dengan memperlihatkan bahwa jika prosedur Huffman mengghasilkan pohon optimal saat $|\Gamma_M| = n - 1$, prosedur Huffman tersebut tidak dapat mengghasilkan pohon optimal saat $|\Gamma_M| = n$.

Pertimbangkan keadaan dimana $|\Gamma_M|$ sama dengan n, dan asumsikan prosedur `Huffman` diatas menghasilkan pohon tidak optimal yang akan kita sebut sebagai `T`. Anggap bahwa T_{opt} menjadi pohon pohon optimal dalam keadaan yang sama, dengan `x` dan `y` adalah karakter-karakter dengan frekuensi-frekuensi terkcil pertama dan kedua dalam `M` maka dapat diketahui bahwa `x` haruslah merupakan daun dengan kedalaman maksimum pada T_{opt}, jika tidak seperti itu kita dapat menukar daun ini dengan daun paling rendah pada pohon yang selanjutnya akan mengurangi nilai C_M dan ini merupakan kontradiksi dengan tingkat optimalitas T_{opt}. Lebih jauh, simpul-simpul dalam T_{opt} dapat selalu ditukar tanpa mengakibatkan perubahan apa-apa pada nilai C_M, sehingga simpul-simpul yang menyimpan `x` dan `y` akan menjadi simpul-simpul yang sederajat. Maka, kedua simpul itu juga harus menjadi sederajat pada `T`. Simpul-simpul tersebut akan terpisah pada iterasi pertama dalam prosedur `Huffman`. Sekarang pertimbangkan daun pohon `T'` dan T'_{opt} dihasilkan dengan menggantikan 2 simpul

yang sederajat tadi dengan simpul tunggal yang mengandung karakter z, yang memiliki frekuensi sama dengan $f_M[x] + f_M[y]$. Perhatikan bahwa prosedur Huffman akan menghasilkan T′ dalam keadaan M′ yang didapatkan dengan menggantikan semua kehadiran x dan y dalam M dengan z. Kecuali isinya berupa x dan y, kedalaman setiap karakter pada pohon yang baru tadi adalah sama dengan kedalamannya pada pohon yang lama. Lebih jauh, baik pada T′ maupun T'_{opt}, karakter baru z selalu hadir satu peringkat lebih tinggi dari x maupun y dalam T′ maupun T'_{opt}. Maka, menggunakan persamaan diatas, kita dapat memperlihatkan bahwa:

$$C_{M'}(T'_{opt}) = C_M(T_{opt}) - f_M[x] - f_M[y]$$

dan

$$C_{M'}(T') = C_M(T) - f_M[x] - f_M[y]$$

Karena diasumsikan bahwa $C_M(T_{opt}) < C_M(T)$, maka mengikuti dua persamaan tersebut kita dapat memperlihatkan bahwa $C_{M'}(T'_{opt}) < C_{M'}(T')$. Tetapi, T′ adalah pohon Huffman dimana $|\Gamma| = n - 1$ sehingga ketidaksamaan sebelumnya memiliki kontradiksi dengan hipotesis induksi yang telah dijabarkan. Kontradiksi ini mengakibatkan prosedur Huffman tidak dapat menghasilkan pohon tidak optimal saat $|\Gamma| = n$, jika prosedur Huffman menghasilkan pohon optimal saat $|\Gamma| = n - 1$. Hal ini memverifikasi langkah-langkah perluasan yang telah dijabarkan sebelumnya dan lebih jauh membuktikan bahwa prosedur Huffman menghasilkan pohon optimal untuk setiap keadaan.

4. Kesimpulan

Pemampatan data dengan algoritma Huffman dalam pengerjaannya memiliki efektifitas yang tergolong tinggi. Teknik pemampatan data dengan algoritma Huffman ini mampu memberikan penghematan sebesar 20% sampai 30%.

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

Kode Huffman sendiri pada dasarnya merupakan kode prefiks (*prefix code*). Kode prefiks adlah himpunan yang berisi sekumpulan kode biner yang dalam hal ini tidak ada kpode biner yang merupakan awal dari kode biner yang lain.

Secara garis besar, langkah-langkah yang dilakukan dalam 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 *gsreedy* 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
4. Baca kembali karakter-karakter di dalam data, kodekan setiap karakter dengan kode Huffman yang bersesuaian.

Biaya (*cost*) penggabungan dua buah pohon pada sebuah akar setara dengan jumlah frekuensi dua buah pohon yang digabung. Oleh karena itu, total biaya pembentukan pohon Huffman adalah jumlah biaya seluruh penggabungan. Penggabungan dua buah pohon dilakukan pada setiap langkah dan algoritma Huffman selalu memilih dua buah pohon yang mempunyai frekuensi terkecil untuk meminimasi total biaya. Hal tersebutlah yang mengakibatkan strategi penggabungan dua yang mempunyai frekuensi terkecil adalah strategi *greedy*. Algoritma Huffman mempunyai kompleksitas O(n log n) untuk himpunan dengan n karakter.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. (2006). Diktat Kuliah IF2153 Matematika Diskrit. Departemen Teknik Informatika, Institut Teknologi Bandung.

[2] Munir, Rinaldi. (2006). Diktat Kuliah IF2251 Strategi Algoritmik. Departemen Teknik Informatika, Institut Teknologi Bandung.

[3] Nugroho, Adi. (2004). Pemrograman Berorientasi Objek. Informatika.

[4] Liem, Inggriani. (2001). Diktat Kuliah IF2181 Struktur Data. Departemen Teknik Informatika, Institut Teknologi Bandung.

[5] Wikipedia, The Free Encyclopedia. (2006), *http://www.wikipedia.org*. Tanggal akses: 27 Desember 2006 pukul 20:00.