PEMBUATAN PROGRAM KOMPRESI DENGAN METODE PENGKODEAN HUFFMAN

Jap Wie Tjhe[1]
[1] Teknik Informatika, STMIK Widuri
[1]Jl. Palmerah Barat No. 353, Jakarta Selatan 12210, Indonesia
Email:[1]*wietjhe@gmail.com*

Abstrak

Sebuah metode kompresi yang cukup terkenal adalah Pengkodean Huffman. Metode ini mengganti data dengan sebuah kode yang disebut dengan kode Huffman. Kode ini didapat dari traversal binary tree. Kode tersebut disimpan dalam file sebagai header dari file. Data yang dikompress disimpan setelah header. Penelitian ini membahas bagaimana cara menyimpan kode Huffman dan data yang sudah terkompress ke sebuah file dan bagaimana cara mendekompresnya.

Kata kunci—*Metode Kompressi dan Dekompresi, Pengkodean Huffman, Struktur Data*

Abstract

A well known compression method is Huffman Coding. This method replace data with a code called Huffman code. The code be created from binary tree traversal. The code is saved in a file as header file. Compressed data is saved after header file. This research discuss how to save Huffman Code and compressed data into a file and how to decompress it.

Keywords—*Compress and Decompress Method, Huffman Coding, Data Structure*

1. PENDAHULUAN

Kompresi adalah proses peringkasan data sehingga membutuhkan memori yang lebih sedikit untuk penyimpanan data tersebut. Kompresi terdapat dua jenis yaitu kompresi *lossless* dan kompresi *lossy*. Kompresi *Lossless* adalah kompresi yang ketika file hasil kompresi didekompress (dikembalikan ke asal) dapat kembali dengan sempurna. Kompresi *Lossy* adalah kompresi yang ketika file hasil kompresi didekompress tidak dapat kembali dengan sempurna [1].

Kode Huffman adalah salah satu metode kompressi jenis *lossless*. Prinsip dari kode Huffman adalah mengganti komponen data pembentuk file dengan kode biner. Komponen data yang sering muncul akan diberi kode biner yang sedikit kode bitnya, sementara data yang jarang muncul akan diberi kode biner lebih panjang. Maka terjadi pengurangan kebutuhan memori untuk menyimpan data [2].

Komponen data yang sering muncul atau jarang muncul dapat diketahui dengan menghitung komponen data tersebut di dalam file. Selanjutnya dilakukan pengurutan secara *descending* berdasarkan banyaknya kemunculan. Melalui frekuensi kemunculan tersebut kemudian dibuat pohon biner. Kode Huffman didapat dari *traversal* pohon biner tersebut.

Pohon biner hanya dapat dibuat dengan menggunakan bahasa komputer yang memiliki kemampuan menangani struktur data. Sementara itu, pohon biner dalam bekerjanya sangat tergantung dengan alokasi RAM yang disediakan oleh sistem operasi, sementara RAM memiliki kemampuan menyimpan yang sangat terbatas [3].

Kode Huffman dapat ditentukan dengan menggunakan larik dua dimensi tanpa harus menggunakan pohon biner. Kode Huffman yang didapat perlu diorganisasi penyimpanannya dalam file hasil kompresi. Organisasi penyimpanan ini nantinya yang dijadikan acuan ketika melakukan dekompres. Komponen data yang akan dicari kode Hufmannya dapat berupa karakter atau kata. Adapun dalam penelitian ini akan digunakan komponen data karakter.

Di sisi lain, larik dua dimensi [3] dapat dengan mudah dibawa ke bentuk tabel basis data. Larik dua dimensi memang disimpan dalam RAM yang memiliki kemampuan menyimpan yang terbatas, namun apabila bentuk larik diterjemahkan ke bentuk tabel basis data, maka tidak ada lagi masalah keterbatasan penyimpan data.

Kebutuhan akan kompresi data saat ini bukan hanya di bidang komunikasi, melainkan di bidang keilmuan lainnya. Adanya penemuan IoT (*Internet of Things*) untuk pembelajaran mesin contohnya ada kebutuhan untuk mengkompresi data, maka data yang diterima lebih banyak dan waktu yang dibutuhkan dalam pengerjaannya juga demikian. Maka pelatihan dibutuhkan seperti pada Jaringan Saraf Tiruan banyak data yang harus diolah dan dilatih melalui interval waktu secara kecil-kecilan dengan waktu kompresi akan sangat membantu, karena ada kebutuhan untuk memproses data lebih cepat dan lebih cepat lagi [4].

Keuntungan dari melihat kebelakang ini sangat sederhana, yang merupakan ide Huffman. Maka n simbol dimasukkan secara alfabet dan digunakan sebagai bobot awal-awal yang melekat pada satu set simpul daun, yang merupakan satu per simbol alfabet. Proses selanjutnya diterapkan melalui dua catatan berbobot paling rendah diidentifikasikan dan dihapus himpunan, lalu digabungkan agar membuat simpul internal baru yang diberikan bobot yaitu merupakan jumlah dari bobot kedua komponennya. Melalui pasangan simpul-bobot baru tersebut lalu ditambahkan kembali ke set dan proses diulang [5].

Algoritma Huffman yang digunakan mengikuti pemecahan masalah *metaheuristic* agar pemilihan menjadi optimal, yang merupakan algoritma serakah. Solusi optimal terselesaikan, melalui cara menghitung setiap langkah. Contohnya menerapkan strategi serakah TSP (*Travelling Salesman Problem*) untuk mengunjungi tempat terdekat yang belum dikunjungi. Algoritma serakah ini tidak pernah menemukan solusi global, tetapi algoritma ini bagus dalam nemenukan solusi global [6].

Kompresi data lossless populer seperti DEFLATE dan GZIP menggunakan algoritma dengan metode pengkodean Huffman merupakan alat utama untuk kompresi. Peran penting yang memainkan dalam pemrosesan sinyal adalah kompresi data atau kompresi sinyal. Penelitian ini memberikan tujuan untuk menjelaskan metode pengkodean Huffman untuk kompresi data *lossless*, dan fungsinya ditunjukkan menggunakan alat MATLAB dan simulasi dilakukan melalui perangkat lunak simulasi dengan program yang ditulis dalam VHDL (VHSIC *Hardware Description Language*). Proses *encoding* dan *decoding* Huffman merupakan kebutuhan optimasi, telah memberikan variasi yang luas. Agar mencapai optimalisasi dalam kode yang dihasilkan oleh pembuat enkode Huffman tradisional juga dievaluasi dan dibahas merupakan faktor vital yang dibahas dalam penelitian [7].

Lainnya dari metode Huffman saat ini digunakan untuk Citra MRI (*Magnetic Resonance Imaging*) sebagai benda universal untuk pemeriksaan dalam kedokteran modern. Metode Huffman ini dapat membantu dokter untuk menganalisis kondisi pasien sedini mungkin. Seperti gambar medis, gambar MRI memiliki kualitas tinggi dan sejumlah besar data, yang membutuhkan lebih banyak waktu transmisi dan kapasitas penyimpanan yang lebih besar. Agar dapat mengurangi waktu transmisi dan kapasitas penyimpanan, kompresi dan teknologi dekompresi diterapkan. Sekarang sebagian besar gambar MRI warna, tetapi sebagian besar penelitian yang dilakukan masih menggunakan gambar MRI abu-abu. Gambar MRI berwarna terkompresi adalah bidang penelitian baru [8].

2. METODE PENELITIAN

Buku Pengolahan Citra dengan Pendekatan Algoritmik karangan Rinaldi Munir, secara garis besar cara menentukan kode Huffman. Maka ditentukan Pengkodean Huffman adalah sebagai berikut: [2]

1. Kumpulkan komponen data pembentuk file.
2. Hitung berapa kali data tersebut muncul di dalam file tersebut.
3. Lakukan pengurutan data pembentuk berdasarkan frekuensi kemunculannya secara *ascending*.
4. Buatlah *binary tree* untuk data pembentuk tersebut berdasarkan frekuensi kemunculannya.
5. Lakukan *traversal* secara secara *in-order*, langkah *traversal* dicatat, ketika *traversal* bergerak ke cabang kiri pohon catat sebagai 0 dan bergerak ke kanan cabang sebagai 1.
6. Kode Huffman didapat dari kode *traversal*.
   Agar lebih jelas, di sini diberikan contoh kalimat yang akan dicari kode Huffmannya:
   Saya suka makan enak

Disini komponen data pembentuk data kalimat tersebut dapat berupa kata atau berupa karakter. Di penelitian ini ditentukan komponen data pembentuk kalimat adalah karakter.

Langkah pertama adalah melakukan statistik frekuensi kemunculan karakter, hasilnya adalah: S(1), a(6), y(1), spasi(3), s(1), u(1), k(3), m(1), n(2), e(1).

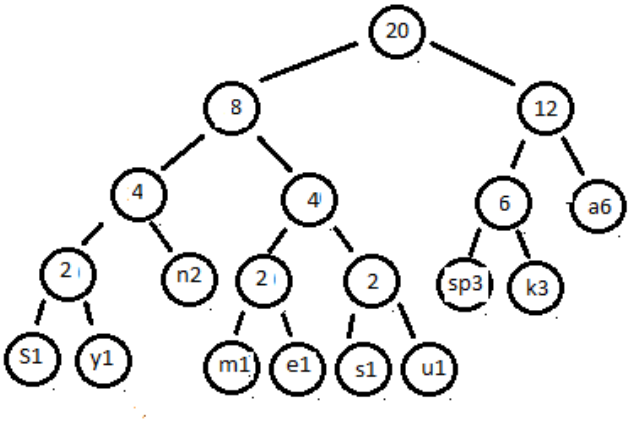
Kemudian dilakukan pengurutan frekuensi sehingga didapat : S(1), y(1), s(1), u(1), m(1), e(1), n(2), spasi(3), k(3), a(6).

Pengkodean menggunakan Huffman tidak akan bingung dalam *decoding*. Cara terbaik untuk melihat hal ini adalah dengan membayangkan *decoder* berputar melalui struktur pohon, dipandu oleh bit yang disandikan yang dibacanya, bergerak dari atas ke bawah dan kemudian kembali ke atas. Selama bit merupakan kode Huffman yang sah, dan bit tidak diacak atau hilang, maka *decoder* juga tidak akan pernah hilang [9]. Kode dengan panjang variabel yang ditetapkan untuk karakter masukan adalah Kode Awalan, artinya kode (urutan bit) ditetapkan sedemikian rupa sehingga kode yang ditetapkan untuk satu karakter bukan awalan kode yang ditetapkan untuk karakter lain mana pun. Cara Huffman membuat sandi adalah memastikan bahwa tidak ada ambiguitas saat mendekode *bitstream* yang dihasilkan. Peneliti dapat memahami bahwa kode awalan dengan contoh penghitung. Contohnya ada empat karakter a, b, c dan d, dan kode panjang variabel yang sesuai adalah 00, 01, 0 dan 1. Pengkodean ini menyebabkan ambiguitas karena kode yang ditugaskan ke c adalah awalan dari kode yang ditugaskan ke a dan b. Jika aliran bit terkompresi adalah 0001, keluaran yang didekompresi mungkin “cccd” atau “ccb” atau “acd” atau “ab.” Bangun pohon Huffman dari karakter masukan dan lintasi pohon Huffman dan tetapkan kode ke karakter [10]. Sehingga dapat dikatakan bahwa Huffman *encoding* merupakan algoritma di mana kode awalan optimal digunakan untuk mengompresi data tanpa kehilangan informasi. Maka kode awalan berarti urutan bit unik yang ditetapkan ke setiap karakter untuk mencegah ambiguitas saat mendekode aliran bit yang dihasilkan [11].

Berikutnya dibuatkan simpul yang menyimpan informasi karakter, frekuensi kemunculan, *pointer* kiri dan *pointer* kanan. Simpul-simpul ini nantinya akan diorganisasi membentuk pohon biner. Letakkan simpul secara urut *ascending* berdasarkan frekuensinya dalam sebuah tabel. Berikut ini adalah aturan pembuatan pohon biner:

1. Ambil dua simpul paling kiri kemudian gabungkan ke sebuah simpul gabungan. *Pointer* kiri dari simpul gabungan menunjuk ke simpul yang lebih kecil nilai frekuensinya. *Pointer* kanan dari simpul gabungan menunjuk ke simpul yang lebih besar frekuensinya. Jumlahkan nilai frekuensi kedua simpul dan letakkan di frekuensi simpul gabungan.
2. Geser larik simpul sehingga larik elemen ke-2 dan seterusnya berpindah ke elemen 0 dan seterusnya.

3. Letakkan simpul gabungan di larik simpul berdasarkan urutan frekuensinya.
4. Ulangi langkah 1 dan 3 sampai hanya ada 1 elemen simpul di larik.
   Berikut ini adalah bentuk pohon biner contoh kalimat di atas:



Gambar 1 Contoh Pohon Biner Kalimat Saya Suka Makan Enak

5. Lakukan *traversal* secara *inorder*. Selama *traversal*, pergerakan *pointer* menuju *node* terbawah dicatat dengan aturan: Bila pointer kiri dilewati dicatat sebagai 0 dan bila *pointe*r kanan dilewati akan dicatat sebagai 1. Pencatatan *traversal* akan menghasilkan kode yang disebut dengan kode Huffman. Berikut ini adalah kode Huffman untuk pohon biner di atas:
   S: 0000, y: 0001, n: 001, m: 0100, e: 0101, s: 0110, u: 0111, spasi: 100, k: 101, a: 11
   kalimat contoh akan disimpan di file sebagai :

```
0000 11 0001 11 100 0110 0111 101 11 100
  S  a  y   a  sps  s    u    k   a sps
0100 11 101 11 001 100 0101 001 11 101
 m   a  k   a  n  sps  e    n   a  k
```

Kalimat **Saya suka makan enak** ketika disimpan di memori membutuhkan ruang sebesar 20 karater x 1 byte atau 20 byte memori atau 160 bit. Bila kalimat disimpan dengan kode Huffman akan membutuhkan ruang memori sebesar 59 bit. Rasio penghematan memori dihitung dengan: 100% - 59/160 x 100%. Masalah yang didapatkan di sini adalah panjang kode Huffman untuk setiap karakter bervariasi, sementara komputer menyimpan satu karakter selalu dengan 8 bit.

Bahasa komputer yang digunakan untuk membuat kompresi dan dekompresinya pada penelitian ini adalah bahasa Foxpro 2.5. Bahasa ini adalah yang tidak dapat membuat pohon biner, tidak memiliki operator *bitwise* namun memiliki kemampuan menyimpan file biner. Format file kompresi dalam file biner nantinya akan disimpan oleh Foxpro.

3. HASIL DAN PEMBAHASAN

Ada banyak hal yang harus dipikirkan setelah menemukan kode Huffman. Hal-hal tersebut, antara lain:

1. Bagaimana komputer menangani ukuran bit sebuah kode Huffman yang tidak standar. Satuan kelompok bit yang standar dikomputer adalah 8 bit, sementara sebuah kode Huffman dapat memiliki satuan dari 1 bit sampai ke berbagai bit. Kemudian bagaimana data kompresi diorganisasi di dalam file kompresi.
2. Bagaimana menerapkan pohon biner ke bahasa pemrograman dengan menggunakan larik dua dimensi. Pencarian kode Huffman juga dapat dilakukan dengan menggunakan larik dua dimensi. Adapun langkah pencarian kode Huffman dengan menggunakan *larik* dua dimensi adalah:

a. Buatlah larik dua dimensi bernama karakter dengan dimensi baris sebanyak 256 dan dimensi kolom sebanyak dua. Indeks dari baris nantinya mengambil kode ASCII karakter. Kolom ke satu digunakan untuk menyimpan frekuensi kemunculan karakter, sedangkan kolom ke dua digunakan untuk menyimpan nilai karakter.
b. Hitunglah banyaknya elemen larik yang memiliki frekuensi tidak nol misal a elemen.
c. Kemudian buat larik dua dimensi yang bernama huruf dengan banyak baris sebanyak a elemen 3 kolom. Kopikan isi larik karakter ke larik huruf. Elemen *larik* karakter yang dipindahkan adalah elemen yang frekuensinya tidak nol. Nilai kolom 1 karakter masuk ke kolom 1 huruf, nilai kolom 2 masuk ke kolom 2 huruf sedangkan kolom 3 larik huruf nantinya digunakan untuk menyimpan kode Huffman dalam bentuk rangkaian karakter '0' dan '1.'
d. Lakukan pengurutan larik huruf secara *ascending* berdasarkan nilai frekuensi. Karena larik huruf hanya memiliki maksimal 256 elemen, pengurutan dapat dilakukan dengan metode *bubble sort* yang sederhana atau boleh juga metode lain yang lebih rumit.
e. Selanjutnya buatlah lagi *larik* dua dimensi bernama kode dengan elemen baris sebanyak a elemen dan kolom sebanyak 2. Kolom ke 1 digunakan untuk frekuensi, kolom ke 2 digunakan untuk indeks gabungan (fungsinya sama dengan simpul dalam pohon biner).
f. Baca indeks gabungan yang ada di kode baris 1 kolom 2. Baca indeks gabungan yang di list dalam kolom tersebut. Berdasarkan indeks tersebut, tambahkan '0' di larik huruf kolom 3 dengan baris sesuai indeks dari indeks gabungan larik kode. Indeks gabungan larik kode disusun seperti *link list*.
g. Baca indeks gabungan yang ada di kode baris 2 kolom 2. Baca indeks gabungan yang di *list* dalam kolom tersebut. Berdasarkan indeks tersebut, tambahkan '0' di larik huruf kolom 3 dengan baris sesuai indeks dari indeks gabungan larik kode. Indeks gabungan larik kode disusun seperti *link list*.
h. Gabungkan nilai frekuensi larik kode yang ada di kolom 1 baris 1 dengan baris 2. Gabungkan indeks gabungan yang ada di kolom 2 baris 1 dan baris 2 dengan indeks gabungan dari baris 1 berada di depan indeks gabungan baris 2.
i. Geser maju larik kode baris ke 3 sampai frekuensi lebih besar dari frekuensi gabungan baris 1 dan baris 2. Baris 1 larik kode sekarang berisi nilai baris 3, baris 2 larik kode sekarang berisi nilai baris 4 dan seterusnya.
j. Masukkan nilai gabungan di baris berikutnya. Kemudian geser maju baris-baris di belakangnya. Baris terakhir larik kode bagian kolom frekuensi di nolkan dan kolom indeks gabungan diberi tanda "X" yang menyatakan baris tersebut jangan diproses lagi.
k. Ulang langkah 6 sampai nilai frekuensi di baris 1 sama dengan total karakter yang ada di file yang akan dikompres.

Dengan menggunakan contoh kalimat yang diuraikan dalam Bab Dua, didapatkan kode Huffman dengan menggunakan larik sebagai berikut:
S: 0000, y: 0101, n: 001, m: 0110, e: 0001, s: 0111, u: 0100, spasi: 100, k: 101, a: 11
Hasil kode Huffman yang didapat dengan menggunakan pohon biner adalah:
S: 0000, y: 0001, n: 001, m: 0100, e: 0101, s: 0110, u: 0111, spasi: 100, k: 101, a: 11

Terdapat perbedaan hasil antara kode Huffman yang didapat dari pohon biner dengan yang didapat dari larik. Perbedaan tersebut tidak menjadi masalah karena kode Huffman yang berbeda tersebut memiliki ukuran bit yang sama dan frekuensi kemunculan data yang sama.

Pencarian kode Huffman dengan menggunakan larik ini dapat menyelesaikan permasalahan bahasa komputer yang digunakan dalam pembuatan aplikasi kompresi ini. Pohon biner hanya dapat dibuat oleh bahasa komputer yang memiliki kemampuan membuat tipe data

jenis *pointer* dan tipe data komplek jenis struktur, tidak semua bahasa komputer memiliki kemampuan ini.

Kalimat 'Saya suka makan enak' ketika dimasukkan ke larik karakter digambarkan seperti gambar 2. Gambar 2 sampai dengan gambar 9 menggambarkan proses penentuan kode Huffman secara langkah per langkah.

Selanjutnya, isi larik karakter yang frekuensinya tidak 0 disimpan ke larik yang diberi nama larik huruf. Sedangkan larik karakter yang memiliki nilai frekuensi 0 diabaikan. Gambar 2 adalah isi larik huruf yang mendapatkan nilai dari larik karakter. Gambar 3 adalah isi larik huruf dari gambar 2 yang diurutkan berdasarkan frekuensi. Gambar 4 adalah isi larik kode. Indeks gabungan mendapatkan harga awal dari indeks larik huruf.

Berikut ini adalah proses mencari kode Huffman. Proses langkah 1:
Ciptakan dua variabel dengan nama h1 dan h2. Maka h1 dan h2 diberi nilai:
h1 = kode(1, frekuensi) + kode(2, frekuensi) = 1 + 1 = 2
h2 = kode(1,indeks gabungan)+kode(2,indeks gabungan)
h2 = " 0" + " 1" = " 0 1"

Di h2, " 0" berada di kiri, maka di larik huruf kolom Huffman ditambahkan "0." Sementara " 1" berada di kanan, maka di larik huruf kolom Huffman ditambahkan dengan "1." Di larik kode indeks 0 kolom indeks gabungan isinya diganti dengan nilai h2 yaitu " 0 1." Sementara kolom frekuensi diisi dengan h1 yaitu 2. Indeks ke 1 dari larik kode dihapus. Urutkan isi larik kode berdasarkan frekuensi. Hasil akhir larik huruf dan larik kode dapat dilihat di Gambar 5.

Larik Karakter

indeks	Frekuensi	Karakter
0	0	chr(0)
1	0	chr(1)
2	0	chr(2)
sd	Sd	Sd
32	3	Spasi
sd	Sd	Sd
83	1	S
sd	Sd	Sd
97	6	A
sd	Sd	Sd
101	1	E
sd	Sd	Sd
107	3	K
108	0	L
109	1	M
110	2	N
sd	Sd	Sd
115	1	S
116	0	T
117	1	U
sd	Sd	sd
121	1	y
sd	Sd	sd
255	0	chr(255)

Gambar 2. Isi larik karakter setelah pembacaan file

Larik huruf

Indeks	Frekuensi	Karakter	Huffman
0	3	Spasi	
1	1	S	
2	6	a	
3	1	e	
4	3	k	
5	1	m	
6	2	n	
7	1	s	
8	1	u	
9	1	y	

Gambar 3. Isi larik huruf

Proses langkah 2 :
h1 = kode(1, frekuensi) + kode(2, frekuensi) = 1 + 1 = 2
h2 = kode(1,indeks gabungan)+kode(2,indeks gabungan)
h2 = " 2" + " 3" = " 2 3"
Di h2, " 2" berada di kiri, maka di larik huruf kolom Huffman ditambahkan "0." Sementara " 3" berada di kanan, maka di larik huruf kolom Huffman ditambahkan dengan "1." Di larik kode indeks 0 kolom indeks gabungan isinya diganti dengan nilai h2 yaitu " 2 3." Sementara kolom frekuensi diisi dengan h1 yaitu 2. Indeks ke 1 dari larik kode dihapus. Urutkan isi larik kode berdasarkan frekuensi. Hasil akhir larik huruf dan larik kode setelah langkah kedua dapat dilihat di Gambar 7. Setelah proses langkah ketujuh didapat hasil akhir larik huruf dan kode seperti Gambar 8.

Kode Huffman yang didapat dari kalimat contoh akan disimpan dalam ukuran byte. Kode Huffman akan disimpan dibagian *higher order bit* dalam sebuah byte, yaitu diletakkan di bagian kiri menuju kanan byte. Sisa ruang byte di bagian kanan yang tidak digunakan kode Huffman akan diisi dengan 0. Kemudian byte selanjutnya akan menyimpan panjang bit kode Huffman yang sebenarnya. Gambar 9 adalah kode Huffman yang didapat dari pohon biner di atas saat disimpan ke file.

Larik huruf urut frekuensi

indeks	frek	Karakter	Huffman
0	1	S	
1	1	E	
2	1	M	
3	1	S	
4	1	U	
5	1	Y	
6	2	N	
7	3	Spasi	
8	3	K	
9	6	A	

Gambar 4. Isi larik huruf urut frekuensi

Larik kode

indeks	Frek	Indeks gabungan
0	1	" 0"
1	1	" 1"
2	1	" 2"
3	1	" 3"
4	1	" 4"
5	1	" 5"
6	2	" 6"
7	3	" 7"
8	3	" 8"
9	6	" 9"

Gambar 5. Isi larik kode

S : 00000000 = 00H panjang = 04H. e: 00010000 = 10H panjang = 04H
m: 01100000 = 60H panjang = 04H. s: 01110000 = 70H panjang = 04H
u : 01000000 = 40H panjang = 04H. y: 01010000 = 50H panjang = 04H
n : 00100000 = 20H panjang = 03H. spasi: 10000000 = 80H panjang = 03H
k : 10100000 = A0H panjang = 03H. a: 11000000 = C0H panjang = 02H

Byte tempat menaruh kode Huffman ada yang diberi garis bawah. Bit-bit yang diberi garis bawah adalah kode Huffman yang sebenarnya. Kode Huffman kemudian diletakkan di *header* secara urut *ascending* menurut kode Huffman untuk memudahkan pencarian kode saat dekompres. Program dekompres membaca kalimat yang sudah dikompres secara bit per bit. Ketika membaca satu bit, program akan mencocokkan dengan daftar kode Huffman yang ada di *header*, selama ditemukan bit yang sesuai dan panjang bit yang dicocokkan belum sampai panjang bit kode Huffman, maka program akan membaca lagi bit berikutnya dan menggabungkan dengan bit tersebut dengan bit sebelumnya sebagai *bit lower order*. Jika panjang bit yang dicari sama dengan panjang bit kode Huffman, baca nilai karakter tersebut kemudian tambahkan ke string hasil dekompres. *Reset* kembali bit untuk mencari kode Huffman, lalu ulangi lagi pencarian seperti sebelumnya. File tempat menyimpan *header* dan hasil kompresi disimpan ke file dengan ekstensi .huf. Gambar 8 adalah isi file .huf yang merupakan hasil kompresi dengan kode huffman:

Array huruf

index	frekuensi	Karakter	Huffman
0	1	S	"0"
1	1	E	"1"
2	1	M	
3	1	S	
4	1	U	
5	1	Y	
6	2	N	
7	3	Spasi	
8	3	K	
9	6	A	

Array kode

index	frekuensi	Index gabungan
0	1	" 2"
1	1	" 3"
2	1	" 4"
3	1	" 5"
4	2	" 6"
5	2	" 0 1"
6	3	" 7"
7	3	" 8"
8	6	" 9"
9	0	"X"

Gambar 6. Isi Larik Huruf dan Kode Setelah Langkah 1

Array huruf

index	frekuensi	karakter	huffman
0	1	S	"0"
1	1	E	"1"
2	1	M	"0"
3	1	S	"1"
4	1	U	
5	1	Y	
6	2	N	
7	3	Spasi	
8	3	K	
9	6	A	

Array kode

index	frekuensi	Index gabungan
0	1	" 4"
1	1	" 5"
2	2	" 6"
3	2	" 0 1"
4	2	" 2 3"
5	3	" 7"
6	3	" 8"
7	6	" 9"
8	0	"X"
9	0	"X"

Gambar 7. Isi Larik Huruf dan Kode Setelah Langkah 2

Array huruf

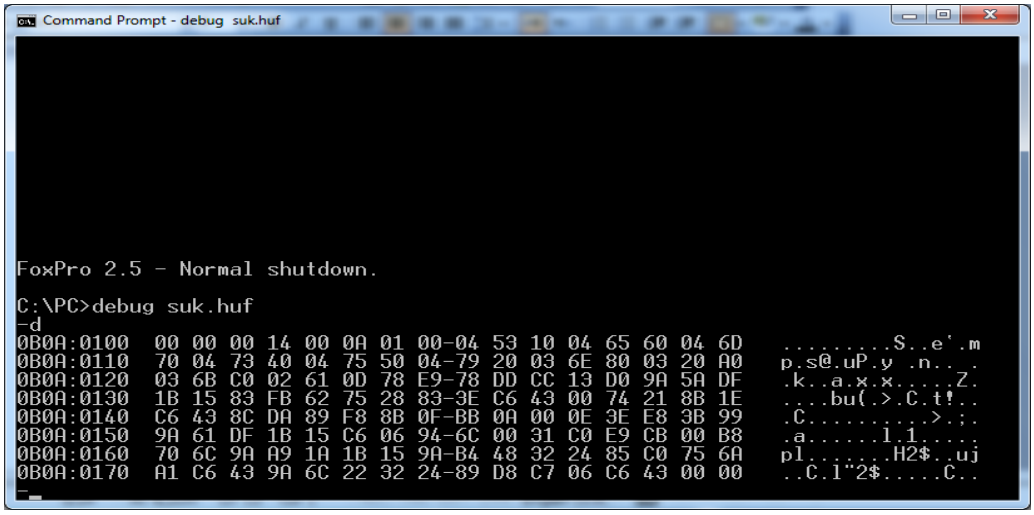
index	frekuensi	karakter	huffman
0	1	S	"0"
1	1	E	"1"
2	1	M	"0"
3	1	S	"1"
4	1	U	
5	1	Y	
6	2	N	
7	3	Spasi	
8	3	K	
9	6	A	

Array kode

index	frekuensi	Index gabungan
0	1	" 4"
1	1	" 5"
2	2	" 6"
3	2	" 0 1"
4	2	" 2 3"
5	3	" 7"
6	3	" 8"
7	6	" 9"
8	0	"X"
9	0	"X"

Gambar 8. Hasil Akhir Proses

Dalam Gambar 9 alamat 0100 sampai 0103 berisi informasi banyaknya karakter dari file asal. File asal berisi kalimat ‘Saya suka makan enak’ yang memiliki 20 karakter. Angka 20 dalam hexadesimal adalah 14H yang ditaruh di alamat 0103. Alamat 0104 dan 0105 berisi informasi banyaknya kode Huffman yang disimpan di *header*. Kalimat ‘Saya suka makan enak’ memiliki 10 macam karakter. Setiap jenis karakter memiliki sebuah kode Huffman. Angka 10 diletakkan di alamat 0105 yang merupakan *Least Significant Byte* sebagai 0AH. Alamat 0104 berisi nilai yang merupakan *Most Significant Byte* (MSB).



Gambar 9. Isi File suk.huf yang adalah File Hasil Kompresi

Alamat 0106 berisi banyaknya byte yang digunakan untuk menyimpan kode Huffman. Dari contoh, kode Huffman untuk 10 karakter memiliki panjang dari 2 bit sampai 4 bit yang dapat ditampung dalam 1 byte. Informasi 1 byte disimpan di alamat 0106.

Alamat 0107 berisi 00H yang adalah kode Huffman dari ‘S.’ Alamat 0108 berisi 04H yang menyatakan banyak bit kode Huffman untuk ‘S.’ Alamat 0109 berisi 53H yang adalah ASCII dari karakter ‘S.’ Alamat 010A berisi 10H yang adalah kode Huffman dari ‘e.’ Alamat 010B berisi 04H yang menyatakan banyak bit kode Huffman untuk ‘e.’ Alamat 010C berisi 65H yang merupakan ASCII dari karakter ‘e.’ Dan seterusnya.

Penyimpanan sebuah kode Huffman diperlukan 3 byte, dimana byte pertama untuk kode Huffman, byte kedua untuk panjang bit kode Huffman dan byte ketiga untuk nilai ASCIInya. Karena terdapat 10 macam karakter yang ada di kalimat ‘Saya suka makan enak,’ maka dibutuhkan 10 kali 3 byte alokasi memori di *header*. Setelah penempatan 10 kode Huffman tersebut, selanjutnya disimpan hasil kompresi dari kalimat ‘Saya suka makan enak.’ Penempatan data hasil kompresi diletakkan di alamat 0125. Alamat 0124 adalah tempat terakhir alokasi memori untuk kode Huffman.

Berdasarkan kode Huffman, ‘Saya suka makan enak’ menghasilkan bit-bit kode Huffman sebagai berikut:

0000 11 0101 11 100 0111 0100 101 11 100
0110 11 101 11 001 100 0001 001 11 101

Jika dikelompokkan per 8 bit dari kiri ke kanan akan didapat bilangan biner sebagai berikut:

00001101 01111000 11101001 01111000
11011101 11001100 00010011 1101XXX

Saat dikelompokkan per 8 bit, terdapat sisa 4 bit, untuk menggenapkan jadi 8 bit dibutuhkan 4 bit lagi yang diberi tanda X. Disini X akan diberi nilai 0, sehingga menjadi 11010000. 8 kelompok bit tersebut kemudian diubah ke hexadesimal agar dapat disimpan ke file, sehingga bit-bit di atas menjadi: 0D 78 E9 78 9D CC 13 D0, Bit-bit tersebut dapat dilihat di alamat 0125 sampai 012C di gambar 8.

Berikut adalah algoritma saat *scanning* dekompres:

1. Baca bit data hasil kompres yang berada setelah *header* kode Huffman, kemudian jadikan bit ke 7 (MSB), bit-bit dibelakangnya diisi 0 semua. Catat berapa banyak bit yang sudah dibaca.
2. Cari dalam deretan kode Huffman di atas dengan metode *binary search*.
3. Jika ditemukan cek apakah panjang bit kode Huffman sama dengan banyak bit yang dibaca.
4. Jika tidak sama, baca lagi bit selanjutnya dan tambahkan panjang bit yang dibaca dengan 1 ke langkah 2.
5. Jika sama, ambil karakter asli untuk kode Huffman tersebut, lalu gabungkan ke teks atau string hasil dekompres.
6. Jika belum akhir bit ke langkah 1.
7. Selesai.

Hasil dekompres :

- Harga awal karakter dekompres: 00000000.
- Bit pertama hasil *scanning* kompresi adalah 0, masukkan ke karakter dekompres, maka didapat: 00000000.
- Cari di *header* dan ditemukan sebagai kode karakter S dengan panjang bit kode Huffman = 4.
- Ambil bit ke 2 kemudian tambahkan ke karakter dekompres dan didapat 00000000. Cari di *header* dan ditemukan sebagai kode karakter S.
- Ambil bit ke 3 kemudian tambahkan ke karakter dekompres dan didapat 00000000. Cari di *header* dan ditemukan sebagai kode karakter S.
- Ambil bit ke 4 kemudian tambahkan ke karakter dekompres dan didapat 00000000. Cari di *header* dan ditemukan sebagai kode karakter S.

Karena panjang bit yang dicari sudah sama dengan panjang bit kode Huffman, maka tambahkan karakter S ke String hasil sehingga didapat “S.”

- Jadikan karakter dekompres sebagai 00000000.
- Masukkan bit ke 5 menjadi bit *MSB* karakter dekompres sehingga didapat nilai 10000000.
- Cari kode tersebut di tempat kode Huffman dan didapat spasi dengan panjang bit kode Huffman sama dengan 3.
- Bit ke 6 ditambahkan ke karakter dekompres dan didapat 11000000. Cari di tempat kode Huffman dan didapat a dengan panjang bit kode Huffman sama dengan 2.

Karena panjang bit yang dicari sudah sama dengan panjang bit kode Huffman, maka tambahkan karakter a ke String hasil sehingga didapat “Sa.”

- Jadikan karakter dekompres sebagai 00000000 dan seterusnya. Tabel 1 adalah sampel dari hasil kompres dan dekompres.

Tabel 1. Hasil Kompres dan Dekompres

Nama File	Ukuran Awal(byte)	Ukuran hasil kompresi(byte)	Rasio(%)
Coba2.doc	24.064	5.347	78
Hufc.doc	24.576	6.362	74
Putih.xls	21.504	11.931	45
As.txt	20	45	-56

Ketika dilakukan peng-kompres-an sembarang file, maka didapatkan hasil:

- *Software* kompresi berhasil mengkompres sembarang file.
- *Software* dekompres hanya berhasil mendekompres beberapa file MS Word dan Excell yang hanya berisi teks. Sementara jika di dalam file tersebut disisipkan gambar, maka *software* dekompres gagal mendekompresnya.
- Kegagalan terjadi karena terdapatnya *bug* di dalam *software* dekompres. Diduga *bug* terjadi ketika menterjemahkan bit kode Huffman terjadi kesalahan reset.
- File yang berisi data gambar akan gagal didekompres.
- Kecepatan kompres dan dekompres lambat.
- Rasio file awal dengan file hasil kompres cukup signifikan, yaitu diatas 50%. Namun ini terjadi hanya untuk file ukuran besar.
- File ukuran kecil ketika dikompres menghasilkan ukuran yang lebih besar dari file asalnya. Ini disebabkan ukuran header lebih besar dari ukuran data terkompresnya.

4. KESIMPULAN

1. Pencarian kode Huffman akan selalu dilakukan setiap kali terdapat perubahan isi file. Ini terjadi karena perubahan isi file akan menyebabkan terjadi perubahan statistik karakter. Namun demikian, cara menyimpan kode Huffman ke file hasil kompresi tetap sama, yaitu diletakkan di *header* file. Demikian pula cara dekompres juga akan sama, yaitu menterjemahkan isi kompresi berdasarkan kode Huffman yang ada di *header* file.
2. Dalam metode di atas, komponen data pembentuk kalimat adalah karakter, sebenarnya komponen data pembentuk kalimat dapat pula berbentuk kata. Hasil kompresi antara komponen data karakter dan komponen data kata akan menghasilkan kompresi yang berbeda.
3. Jika file yang dikompres berukuran kecil akan mengakibatkan ukuran file hasil kompress lebih besar dari file aslinya. Efisiensi memori dapat dirasakan apabila file yang dikompress berukuran besar dan banyak sekali terjadi pengulangan karakter. File citra dapat dikompress dengan mengasumsikan file tersebut adalah file teks.
4. Pencarian kode Huffman dengan *traversal* pohon biner dapat digantikan dengan baik dengan menggunakan larik dua dimensi.

DAFTAR PUSTAKA

[1] Pangesti, Witriana Endah dkk, 2020, *Implementasi Kompresi Citra Digital dengan Membandingkan Metode Lossy dan Lossless Compression Menggunakan MATLAB*, Jurnal Khatulistiwa Informatika, Vol. VIII, No. 1, hal. 53-58.

[2] Munir, Rinaldi, 2004, *Pengolahan Citra Digital – dengan Pendekatan Algoritmik*, Informatika, Bandung.

[3] B, Yatini, Indra, dan Nasution, Erliansyah, 2005, *Algoritma dan Struktur Data dengan C++*, Graha Ilmu, Yogyakarta.

[4] R, Gautam dan Murali, S, 2016, *An Optimized Huffman's Coding by the method of Grouping*, Computer Science, Information Theory, Cornell University, Ithaca, New York, hal. 1-4, July 23.

[5] Moffat, Alistair, 2019, *Huffman Coding*, ACM Comput. Surv., Association for Computing Machinery, New York, Vol. 1, No. 1, Article 1, hal. 1-35.

[6] Suherman dan Siahaan, Andysah Putera Utama, 2016, *Huffman Text Compression Technique*, SSRG International Journal of Computer Science and Engineering (SSRG-IJCSE), Vol. 3, Issue 8, hal. 103-108.

[7] Dhawale, Nidhi, 2014, *Implementation of Huffman algorithm and study for optimization*, International Conference on Advances in Communication and Computing Technologies (ICACACT), IEEE Xplore, August 10-11.
[8] Zou, Xin, 2018, *Compression and Decompression of Color MRI Image by Huffman Coding*, Department of Electrical Engineering Blekinge Institute of Technology, Karlskrona, Sweden, hal. 1 – 64.
[9] Shoba, D. Jasmine dan Sivakumar, S., 2017, *A Study on Data Compression Using Huffman Coding Algorithms*, International Journal of Computer Science Trends and Technology (IJCST) – Volume 5, Issue 1, hal. 58-63, Jan – Feb.
[10] Kddeepak, dkk. 2023, Huffman Coding | Greedy Algo-3.
[11] Experts, UK Essay, 2021, *What is Huffman Encoding and why is it important?*, Business Bliss Consultants FZE, Creative Tower, Fujairah, October 18.