IMPLEMENTASI METODE HUFFMAN UNTUK KOMPRESI UKURAN FILE CITRA BITMAP 8 BIT MENGGUNAKAN BORLAND DELPHI 6.0

Rina Dewi Indah Sari, S. Kom
Nur Munawaroh

ABSTRAKSI

Algoritma metode Huffman adalah salah satu algoritma kompresi untuk citra digital. Metode kompresi Huffman menggunakan prinsip bahwa nilai derajat keabuan yang sering muncul di dalam citra akan dikodekan dengan jumlah bit yang lebih sedikit sedangkan nilai keabuan yang frekuensi kemunculannya sedikit akan dikodekan dengan jumlah bit yang lebih panjang. Citra yang mempunyai sebaran nilai piksel tidak merata memiliki rasio kompresi yang relatif besar sedangkan citra dengan nilai piksel yang merata memiliki rasio kompresi yang lebih kecil.
Kata kunci : Citra, Kompresi, algoritma Huffman, pohon Huffman

I. PENDAHULUAN

1. Latar Belakang

Data atau informasi tidak hanya disajikan dalam bentuk teks, tetapi juga dapat berupa gambar, *audio* (bunyi, suara, musik) dan video. Keempat macam data atau informasi ini sering disebut dengan multimedia. Era informasi teknologi saat ini tidak dapat dipisahkan dari multimedia. Situs web *(website)* di internet dibuat semenarik mungkin dengan menyertakan visualisasi berupa gambar atau video yang dapat diputar [1].

Pada umumnya representasi citra digital membutuhkan memori yang besar. Semakin besar ukuran citra tentu semakin besar pula memori yang dibutuhkannya. Pada sisi lain, kebanyakan citra mengandung duplikasi data. Duplikasi data pada citra dapat berarti dua hal. Pertama, besar kemungkinan suatu *pixel* dengan *pixel* tetangganya memiliki intensitas yang sama, sehingga penyimpanan setiap *pixel* memboroskan tempat. Kedua, citra banyak mengandung bagian *(region)* yang sama, sehingga bagian yang sama ini tidak perlu dikodekan berulang kali karena redundan .

Saat ini, kebanyakan aplikasi menginginkan representasi citra dengan kebutuhan memori yang sesedikit mungkin. Kompresi citra *(image compression)* bertujuan meminimalkan kebutuhan memori untuk merepresentasikan citra digital. Prinsip umum yang digunakan dalam kompresi citra adalah mengurangi duplikasi data di dalam citra sehingga memori yang dibutuhkan untuk merepresentasikan citra menjadi lebih sedikit dari pada representasi citra semula.

Kompresi citra memberikan manfaat yang sangat besar dalam industri multimedia saat ini. Salah satunya adalah pada proses pengiriman data *(data transmission)* pada saluran komunikasi data. Citra yang telah dikomprsesi membutuhkan waktu pengiriman yang lebih singkat dibandingkan dengan citra yang tidak dikompresi. Contohnya aplikasi pengiriman gambar lewat *fax, videoconferencing*, pengiriman data medis, pengiriman gambar dari satelit luar angkasa, pengiriman gambar via telepon genggam, *download* gambar dari internet, dan sebagainya. Selain pada proses pengiriman data, kompresi citra juga bermanfaat pada penyimpanan data *(data storing)* di dalam media sekunder *(storage).* Citra yang telah dikompresi membutuhkan ruang memori di dalam media *storage* yang lebih sedikit dibandingkan dengan citra yang tidak dikompresi. Contoh aplikasinya antara lain aplikasi basisdata gambar, *office automation, video storage (*seperti *Video Compact Disc)* [1].

Ada dua tipe utama kompresi citra, yaitu kompresi tipe *lossless* dan kompresi tipe *lossy*. Kompresi tipe *lossy* adalah kompresi dimana terdapat data yang hilang selama proses kompresi.

[1] Rinaldi Munir,*Pengolahan Citra Digital dengan Pendekatan Algoritmik*(Bandung:Informatika,2004,hal.1

Akibatnya kualitas citra yang dihasilkan jauh lebih rendah daripada kualitas citra asli. Sementara itu, kompresi tipe *lossless* tidak menghilangkan informasi setelah proses kompresi terjadi, akibatnya kualitas citra hasil kompresi tidak menurun. Salah satu contoh dari kompresi tipe *lossless* adalah metode Huffman. Metode Huffman paling efisien dari metode lain yang sejenis karena pemetaan lain simbol dari sumber data menjadi string unik menghasilkan file *output* yang lebih kecil ketika frekuensi symbol sesuai dengan frekuensi yang digunakan untuk menghasilkan kodenya [2].

2. Rumusan Masalah

1. Bagaimana cara kerja dari Metode Huffman sebagai teknik kompresi citra
2. Bagaimana tingkat efisiensi memori file hasil kompresi dengan menggunakan Metode Huffman.
3. Bagaimana implementasi dari Metode Huffman sebagi teknik kompresi citra dengan menggunakan bahasa pemrograman Borland Delphi 6.0.

3. Batasan Masalah

1. Citra yang digunakan adalah citra *gray scale* dalam format BMP dengan kedalaman warna 8 bit.
2. Metode Huffman diimplementasikan menggunakan Borland Delphi 6.0.

4. Tujuan

1. Untuk mengetahui dan memahami cara kerja teknik kompresi citra dengan metode Huffman.
2. Untuk mengetahui tingkat efisiensi memori file hasil kompresi citra dengan Metode Huffman.
3. Untuk mengimplementasikan Metode Huffman sebagai teknik kompresi citra dengan menggunakan Borland Delphi 6.0..

5. Metode Penelitian

1. Studi Pustaka (*Library Research*)

   Studi Pustaka dilakukan dengan cara mempelajari teori-teori literatur dan buku-buku yang berhubungan dengan objek kajian sebagai dasar dalam penelitian, dengan tujuan memperoleh dasar teoritis gambaran dari apa yang dilakukan.

TINJAUAN PUSTAKA

1. Representasi Citra Digital

a. Citra Biner (monokrom)

Pada citra biner setiap titik bernilai 0 atau 1, masing-masing merepresentasikan warna tertentu. Contohnya adalah warna hitam bernilai 0 dan warna putih bernilai 1, pada standar citra yang ditampilkan di layar komputer, nilai biner ini berhubungan dengan ada tidaknya cahaya yang ditembakkan oleh *electron gun* yang terdapat dalam monitor komputer. Angka 0 menyatakan tidak ada cahaya, dengan demikian warna yang direpresentasikan adalah hitam. Untuk angka 1 terdapat cahaya, sehingga warna yang direpresentasikan adalah putih.

b. Citra Skala Keabuan (*gray scale*)

Citra skala keabuan memberi kemungkinan warna yang lebih banyak daripada citra biner, karena ada nilai-nilai lain diantara nilai minimum (biasanya = 0) dan nilai maksimumnya. Banyaknya kemungkinan nilai dan nilai maksimumnya bergantung pada jumlah bit yang digunakan.

Contohnya dalam skala keabuan 4 bit, maka jumlah kemungkinan nilainya adalah $2^4 = 16$, dan nilai maksimumnya adalah $2^4 – 1 = 15$.

Format citra ini disebuat skala keabuan karena pada umumnya warna yang dipakai adalah antara hitam sebagai warna minimal dan putih sebagai warna maksimal, sehingga warna antaranya adalah abu-abu.

c. Citra Warna (*true colour*)

Pada citra warna setiap titik mempunyai warna yang spesifik yang merupakan kombinasi dari tiga warna dasar, yaitu: merah, hijau, biru. Format citra ini sering disebut sebagai citra RGB (*red-green-blue*). Setiap warna dasar mempunyai intensitas sendiri dengan nilai maksimum 255 (8 bit). Dengan demikian setiap titik pada citra warna membutuhkan 3 *byte.*

Jumlah kombinasi warna yang mungkin untuk format citra ini adalah 2^{24} atau lebih dari 16 juta warna, dengan demikian dianggap mencakup semua warna yang ada, inilah sebabnya format ini dinamakan *true colour.*

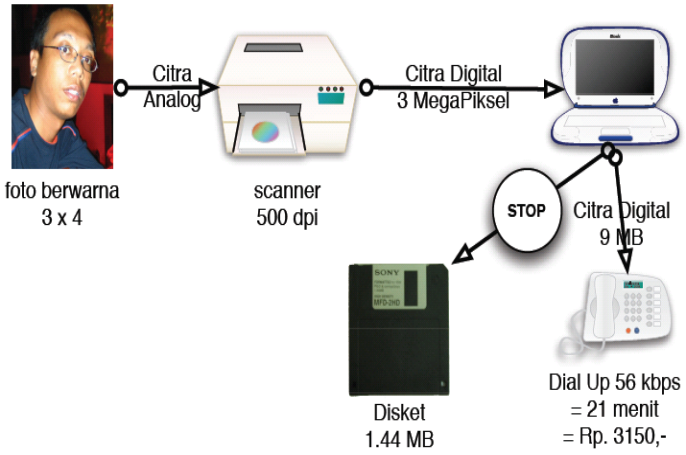
d. Citra Warna Berindeks

Jumlah memori yang dibutuhkan untuk format citra warna *true colour* adalah tiga kali jumlah titik yang ada dalam citra yang ditinjau. Di lain pihak pada kebanyakan kasus, jumlah warna yang ada dalam suatu citra terkadang sangat terbatas (jauh dibawah 10 juta kemungkinan warna yang ada) karena banyaknya warna dalam sebuah citra tidak mungkin melebihi banyaknya titik dalam citra itu sendiri. Untuk kasus tersebut disediakan format citra warna berindeks. Pada format ini, informasi setiap titik merupakan indeks dari suatu tabel yang berisi informasi warna yang tersedia yang disebut palet warna (beberapa buku menyebutnya sebagai *colour map*).

2. Kompresi Citra

Kompresi citra bertujuan untuk meminimalkan jumlah bit yang diperlukan untuk merepresentasikan citra. Apabila sebuah foto berwarna berukuran 3 inci x 4 inci diubah ke bentuk digital dengan tingkat resolusi sebesar 500 *dot per inch* (dpi), maka diperlukan 3 x 4 x 500 x 500 = 3.000.000 dot (piksel). Setiap piksel terdiri dari 3 *byte* dimana masing-masing *byte* merepresentasikan warna merah, hijau, dan biru. sehingga citra digital tersebut memerlukan *volume* penyimpanan sebesar 3.000.000 x 3 *byte* +1080 = 9.001.080 *byte* setelah ditambahkan jumlah *byte* yang diperlukan untuk menyimpan *format* (*header*) citra.

Pengiriman citra berukuran 9 MB memerlukan waktu lebih lama. Untuk koneksi internet *dial-up* (56 kbps), pengiriman citra berukuran 9 MB memerlukan waktu 21 menit. Untuk itulah diperlukan kompresi citra sehingga ukuran citra tersebut menjadi lebih kecil dan waktu pengiriman citra menjadi lebih cepat. Citra yang belum dikompres disebut citra mentah (*raw image)*. Sementara citra hasil kompresi disebut citra terkompresi(*compressed image)*.



Gambar Proses Konversi citra analog ke citra digital dan pengirimannya

Parameter-parameter citra yang penting dalam proses kompresi diantaranya sebagai berikut:

1. Resolusi
   Resolusi citra menyatakan ukuran panjang kali lebar dari sebuah citra. Resolusi citra biasanya dinyatakan dalam satuan piksel. Semakin tinggi resolusi sebuah citra, semakin baik kualitas citra tersebut. Namun, tingginya resolusi menyebabkan semakin banyaknya jumlah bit yang diperlukan untuk menyimpan dan mentransmisikan data citra tersebut.
2. Kedalaman bit
   Kedalaman bit menyatakan jumlah bit yang dipelukan untuk mrepresentasikan tiap piksel citra pada sebuah *frame*. Kedalaman bit biasanya dinyatakan dalam satuan *bit/*piksel. Semakin banyak jumlah *bit* yang digunakan untuk merepresentasikan sebuah citra, maka semakin baik kualitas citra tersebut.
3. Konsep redundansi
   Redundansi merupakan suatu keadaan dimana representasi suatu elemen data tidak bernilai signifikan dalam merepresentasikan keseluruhan data. Keadaan ini menyebabkan data keseluruhan dapat direpresentasikan secara lebih kompak dengan cara menghilangkan representasi dari sebuah elemen data yang redundan. Redundansi yang terdapat pada citra statik adalah redundansi spasial.

Dalam proses kompresi (pemampatan) citra terdapat dua proses utama yaitu sebagai berikut:

1. Pemampatan citra *(image compression)*
   Pada proses, ini citra dalam representasi tidak mampat dikodekan dengan representasi yang meminimumkan kebutuhan memori.
2. Penirmampatan citra *(image decompression)*
   Pada proses ini, citra yang sudah dimampatkan harus dapat dikembalikan lagi *(decoding)* menjadi representasi yang tidak mampat. Proses ini diperlukan jika citra tersebut akan ditampilkan ke layar atau disimpan ke dalam arsip dengan format tidak mampat.

Saat ini sudah banyak ditemukan metode-metode pemampatan citra. Kriteria yang digunakan dalam mengukur metode pemampatan citra adalah [6]:

1. Waktu pemampatan dan penirmampatan *(decompression)*
2. Kebutuhan memori
3. Kualitas pemampatan *(fidelity)*
4. Format keluaran

Ada empat jenis pendekatan dalam pemampatan citra adalah sebagai berikut:

1. Pendekatan statistik
2. Pendekatan ruang

3. Pendekatan kuantisasi
4. Pendekatan *fractal*

Metode pemampatan citra dapat diklasifikasikan ke dalam dua kelompok besar yaitu:

1. Metode *lossless*
   Metode *lossless* selalu menghasilkan citra hasil penirmampatan yang tepat sama dengan citra semula, *pixel* per *pixel.* Tidak ada informasi yang hilang akibat pemampatan. Contoh metode *lossless* adalah metode *Huffman.* Rasio kompresi citra dihitung dengan rumus:

$$\text{Rasio kompresi} = 100\% - \left(\frac{\text{ukuran citra semula}}{\text{ukuran citra hasil pemampatan}} \right) \times 100\%$$

   Metode *lossless* cocok untuk memampatkan citra yang mengandung informasi penting yang tidak boleh rusak akibat pemampatan. Misalnya memampatkan gambar hasil diagnosa medis.
2. Metode *lossy*
   Metode *lossy* menghasilkan citra hasil pemampatan yang hampir sama dengan citra semula. Ada informasi yang hilang akibat pemampatan, tetapi dapat ditolerir oleh persepsi mata. Mata tidak dapat membedakan perubahan kecil pada gambar. Metode pemampatan *lossy* menghasilkan nisbah pemampatan yang tinggi daripada metode *lossless.*

3. Metode Huffman

Dalam ilmu komputer dan teori informasi, kode Huffman adalah algoritma pengkodean untuk kompresi data *lossless.* Istilah ini merujuk kepada penggunaan tabel kode yang memiliki panjang bervariasi (*variable length code*) dimana tabel kode tersebut diturunkan dengan cara tertentu berdasarkan prakiraan probabilitas kemunculan setiap nilai dalam sumber data. Metode ini ditemukan oleh David A. Huffman ketika ia melakukan studi Ph.D di MIT.

Kode Huffman menggunakan metode spesifik untuk merepresentasikan setiap symbol, menghasilkan prefix-free code (*string* dari bit representasi sebuah simbol tidak pernah menjadi prefix (awalan) dari sebuah simbol lain). Yang merepresentasikan karakter yang lebih sering muncul dengan *bit string* yang lebih pendek daripada karakter yang jarang muncul dalam suatu sumber data.

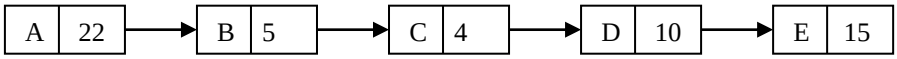
Kompresi Huffman adalah metode paling efisien dari metode lain yang sejenis karena pemetaan lain simbol dari sumber data menjadi string unik menghasilkan file *output* yang lebih kecil ketika frekuensi simbol sesuai dengan frekuensi yang digunakan untuk menghasilkan kodenya.

Kompresi data yang dihasilkan pada algoritma pengkodean Huffman ini dicapai dengan mengkodekan data berdasarkan pada frekuensi kemunculannya. Struktur data yang digunakan untuk mengkodekan data adalah suatu *weighted binary tree,* atau pohon Huffman (*Huffmantree*). Suatu pohon Huffman memiliki ciri-ciri unikn sebagai berikut:

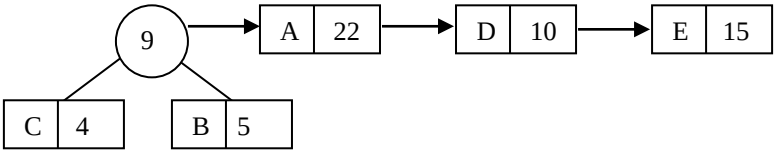
1. Pohon Huffman harus merupakan suatu binary tree.
2. Pada pohon Huffman ini elemen yang paling sering muncul dalam *data stream* berada di puncak pohon sedangkan elemen yang paling jarang muncul berada di dasar pohon.
3. Tiap cabang kiri dari pohon Huffman diberi tanda sebagai nilai nol, dan tiap cabang kanan dari pohon diberi tanda sebagai nilai satu (atau sebaliknya).

Langkah pembentukan pohon Huffman adalah sebagai berikut:

3 Dari semua data yang diketahui, lengkap dengan frekuensi masing-masing data dibentuk satu senarai berantai. Sebagai contoh terdapat lima komponen dengan masing-masing frekuensinya

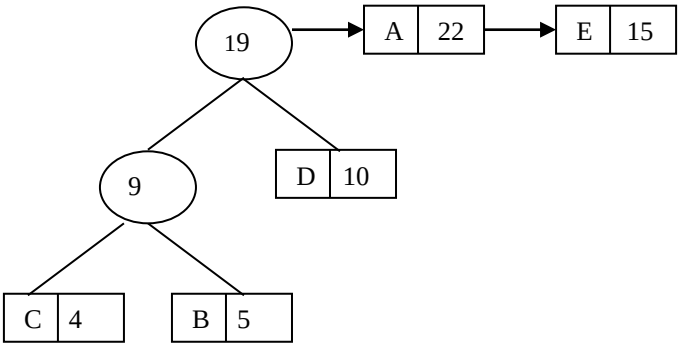


4 Memilih dua buah senarai yang mempunyai frekuensi paling kecil

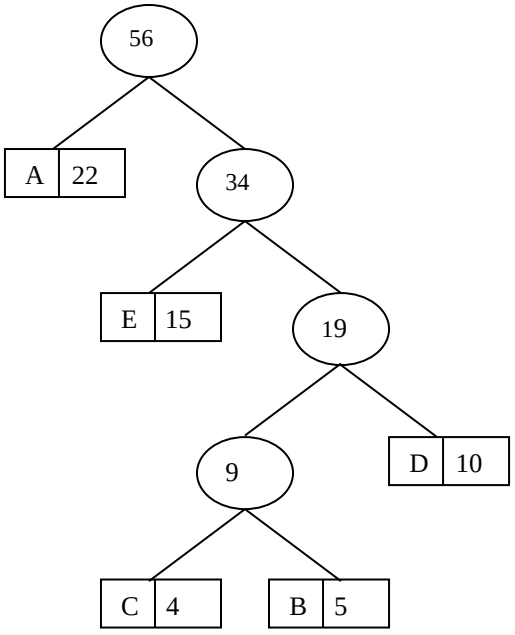


Selanjutnya kedua komponen tersebut kita lepas dari senari berantai. Dari kedua komponen tersebut kita susun sebuah pohon yang akarnya berisi jumlah frekuensi kedua komponen yang dilepas dari senarai berantai tersebut. Selanjutnya akar pohon tersebut kita sisipkan sebagai awal dari senarai berantai.

5 Langkah tersebut diulang sehingga akan diperoleh seperti di bawah ini



6 Ulangi kembali langkah diatas sampai komponen senarai berantai tinggal sebuah yaitu akar pohon secara keseluruhan. Dari setiap dua komponen yang akan dibentuk pohon, komponen yang mempunyai frekuensi lebih kecil dipasang sebagai cabang kiri dan komponen yang lebih besar dipasang di cabang kanan.



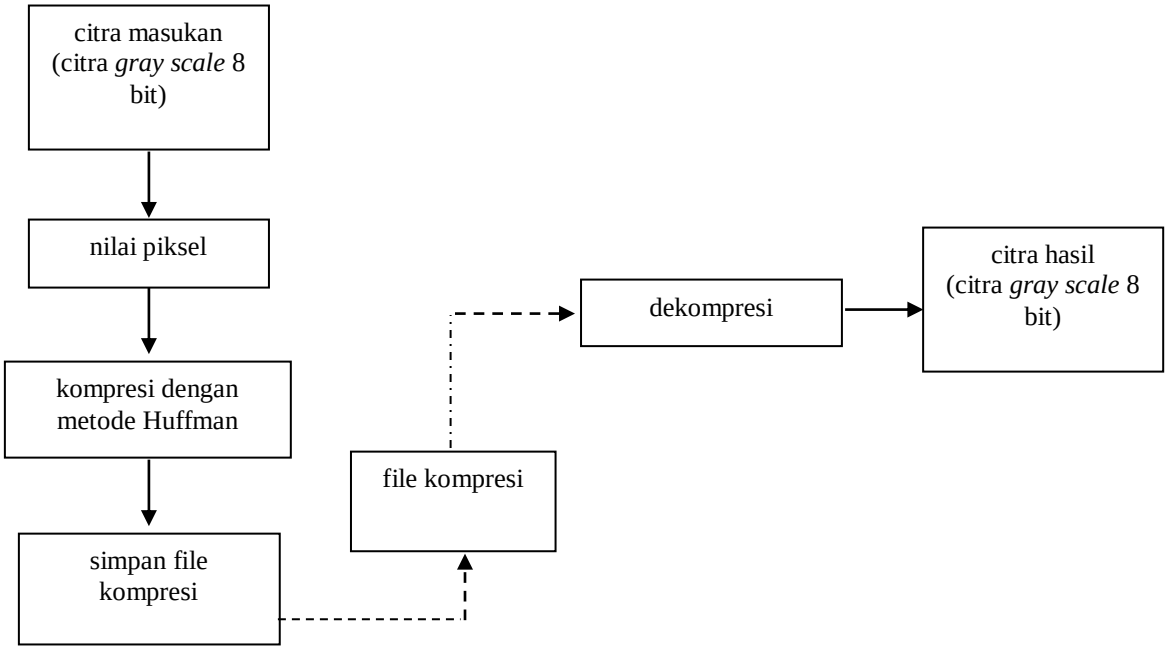
Metode pemampatan Huffman menggunakam prinsip bahwa nilai (atau derajat) keabuan yang sering muncul di dalam citra akan dikodekan dengan jumlah bit yang lebih sedikit sedangkan nilai keabuan yang frekuensi kemunculannya sedikit dikodekan dengan jumlah bit yang lebih panjang.

Algoritma metode Huffman adalah sebagai berikut:

1. Urutkan secara menaik *(ascending order)* nilai-nilai keabuan berdasarkan frekuensi kemunculannya (atau berdasarkan peluang kemunculan, $p_{k,}$ yaitu frekuensi kemunculan (n_k) dibagi dengan jumlah *pixel* di dalam gambar (n). Setiap nilai keabuan dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul di-*assign* dengan frekuensi kemunculan nilai keabuan tersebut.
2. Gabung dua pohon yang mempunyai frekuensi kemunculan paling kecil pada sebuah akar. Akar mempunyai frekuensi yang merupakan jumlah dari frekuensi dua pohon penyusunnya.
3. Ulangi langkah 2 sampai tersisa hanya satu pohon biner.
   Agar pemilihan dua pohon yang akan digabungkan berlangsung cepat, maka semua pohon yang ada selalu terurut menaik berdasarkan frekuensi.
4. Beri label setiap sisi pada pohon biner. Sisi kiri dilabeli dengan 0 dan sisi kanan dilabeli dengan 1.
   Simpul-simpul daun pada pohon biner menyatakan nilai keabuan yang terdapat di dalam citra semula.
5. Telusuri pohon biner dari akar ke daun. Barisan label-label sisi dari akar ke daun menyatakan kode Huffman untuk derajat keabuan yang bersesuaian.

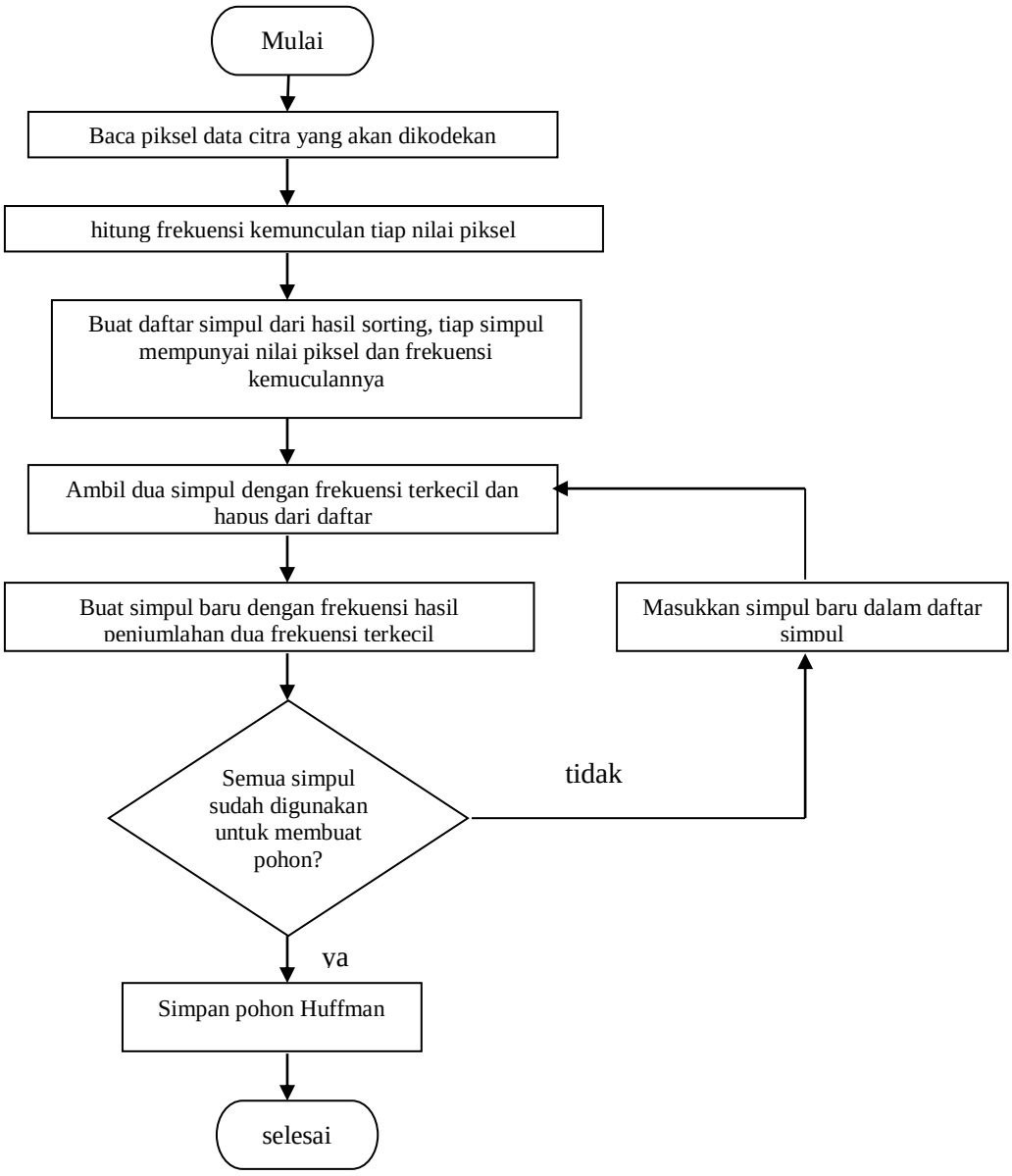
PERANCANGAN

1. Teknik Kompresi



Gambar Teknik Kompresi

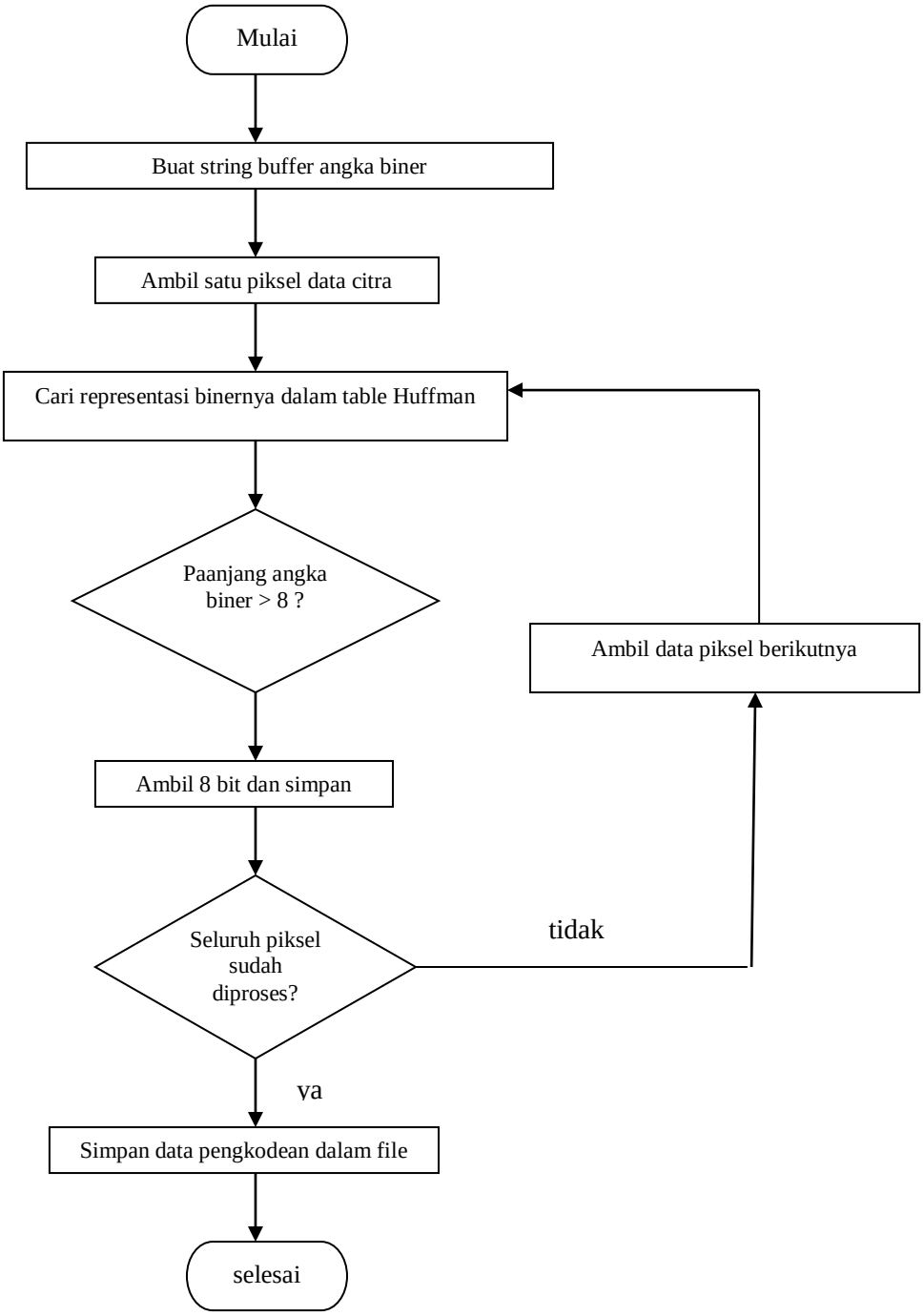
2. Pohon Huffman



Gambar *Flowchart* pembentukan pohon Huffman

Pohon Huffman dibuat dengan menggunakan algoritma pengkodean Huffman di atas. Dari data piksel dihitung frekuensi kemunculannya. Tiap data dianggap satu simpul yang mempunyai dua nilai data: nilai keabuan dan frekuensi kemun-culannya. Dua simpul dengan frekuensi terkecil dari daftar diambil dan frekuensinya dijumlahkan. Hasil penjumlahan kedua frekuensi ini menjadi frekuensi baru untuk sebuah simpul baru yang mempunyai dua cabang kedua simpul tadi (cabang kiri dan kanan). Simpul baru ini selanjutnya dimasukkan dalam daftar untuk dilakukan pengurutan simpul, penjumlahan dua smpul dengan frekuensi terkecil, dan pembuatan simpul baru. Proses ini berlanjut sampai semua simpul telah masuk ke dalam pohon dan frekuensi akar (*root*) dari pohon akan merupakan hasil mpenjumlahan semua frekuensi yang ada.

3. Tabel Huffman



Gambar *Flowchart* pembentukan tabel Huffman

PEMBAHASAN DAN IMPLEMENTASI

1. Komponen TImage pada Delphi

Untuk menampilkan citra yang akan dikompres menggunakan komponen TImage yang terdapat pada palet komponen *Additional* pada Delphi. Komponen ini memiliki properti *Picture* yang digunakan untuk menyimpan data citra. Citra yang akan ditampilkan diambil pada saat program dijalankan dengan menggunakan *procedure LoadFromFile.* Properti yang ada pada picture antara lain:

1. *Height*, berisi nilai tinggi citra.
2. *Width*, berisi nilai lebar citra.
3. *Bitmap*, berisi data format dan piksel citra.

Dalam Delphi, informasi format citra terdapat pada subproperti Bitmap, yaitu PixelFormat. Dengan membaca nilai PixelFormat dapat diketahui cara penyimpanan piksel dalam memori. Sehingga mempermudah dalam pemrograman. Untuk citra *gray scale* 8 bit mempunyai nilai PixelFormat pf8bit, setiap piksel disimpan dalam ukuran 1 byte (8 bit). Pf8bit ini direpresentasikan sebagai citra berindeks 8 bit dengan komponen palet warna merah, hijau dan biru yang bernilai sama, sehingga menampilkan warna keabuan dari hitam sampai putih.

2. Struktur Data

Struktur data yang digunakan pada program kompresi ini menggunakan pointer yang elemennya berupa rekaman (*record*). Ada dua pointer yang digunakan yaitu pointer untuk pembentukan pohon Huffman dan pointer untuk pembentukan kode Huffman.

1. Deklarasi pointer untuk pohon huffman

```
type
 Phuffinfo=^Thuffinfo;
 THuffInfo=record
   kiri:phuffinfo;
   kanan:phuffinfo;
   piksel: array[0..255] of byte;
   jml_kode:integer;
   frek:integer;
   sudah:boolean;
end;
```

Pada deklarasi diatas Phuffinfo menyatakan data yang bertipe pointer. Thuffinfo merupakan nilai data yang ditunjuk oleh pointer Phuffinfo. Simpul-simpul yang akan dibentuk pada pohon Huffman dinyatakan dalam Thuffinfo. Thuffinfo mempunyai elemen sebuah data berupa record (rekaman) yang berisi:

Kiri : bertipe pointer yang menyatakan cabang kiri pohon Huffman.
Kanan : bertipe pointer yang menyatakan cabang kanan pohon Huffman.
Piksel : bertipe array yang menyatakan nilai piksel dari citra yang akan dikompresi.
jml_kode: bertipe integer yang menyatakan jumlah kode dalam akar (root) sebuah pohon huffman.
frek : bertipe integer yang menyatakan frekuensi kemunculan nilai piksel.
sudah: bertipe boolean yang menyatakan apakah nilai piksel yang mempunyai frek > 0 sudah dimasukkan dalam pohon Huffman.

2. Deklarasi pointer untuk kode Huffman

```
type
PhuffCode=^THuffCode;
 THuffcode=record
```

```
    Ada:boolean;
    kode:array[0..255] of byte;
    panjangKode:integer;
  end;
```

Pada deklarasi diatas PhuffCode menyatakan data yang bertipe pointer. THuffCode merupakan nilai data yang ditunjuk oleh pointer PhuffCode. Simpul-simpul kode yang akan dibentuk pada kode Huffman dinyatakan dalam THuffCode. THuffCode mempunyai elemen sebuah data berupa rekaman (*record*) yang berisi:

Ada: bertipe boolean yang menyatakan bahwa apakah nilai piksel dari citra sudah dikodekan
kode: bertipe array yang menyatakan nilai piksel dari kode yang sudah dibentuk
panjangKode: bertipe integer yang menyatakan panjang dari nilai piksel yang sudah dikodekan

3. Teknik Kompresi

Prinsip dari metode Huffman pada citra digital adalah mengkodekan setiap nilai keabuan dengan rangkaian bit 0 dan 1, di mana simbol yang memiliki 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'.

Langkah-langkah yang dilakukan pada proses kompresi adalah sebagai berikut:

1. Menghitung frekuensi kemunculan dari setiap nilai keabuan (piksel) citra yang akan dikompres.

```
if( Fcitra.Image1.Picture.bitmap.PixelFormat= pf8Bit) then
 begin
  for i := 0 to 255 do
   begin
    charlist[i].frek:= 0;
   end;
  x:=0;
  for i := 0 to fcitra.Image1.Picture.Height-1 do
   begin
    PC := fcitra.Image1.Picture.Bitmap.ScanLine[i];
    for j := 0 to fcitra.Image1.Picture.Width-1 do
     begin
      InBuffer[x]:=PC[j];
      Inc(charlist[InBuffer[x]].frek);
      inc(x);
     end;
   end;
```

2. Pembentukan Pohon Huffman

   Langkah-Langkah pembentukan pohon Huffman adalah sebagai berikut:

   1. Pada pembentukan pohon Huffman digunakan dua pointer yaitu kanan (sebagai penunjuk cabang kanan dari pohon Huffman) dan kiri (sebagai penunjuk cabang kiri dari pohon huffman). Selain itu ada juga dua variabel bertipe pointer yaitu pinfo1 dan pinfo2 yang digunakan masing-masing sebagai simpul dari setiap cabang.

2. Pembentukan pohon Huffman dilakukan dengan membuat simpul (charlist) yang mempunyai nilai data yaitu nilai piksel dan frekuensi kemunculannya (charlist.frek).
3. Dicari simpul yang memiliki frekuensi terkecil pertama, kemudian disimpan dalam pinfo1.frek. Selanjutnya mencari simpul yang memiliki frekuensi terkecil kedua dan disimpan dalam pinfo2.frek.
4. Dari dua simpul dengan frekuensi terkecil tadi dijumlahkan frekuensinya.
5. Hasil penjumlahan kedua frekuensi ini menjadi frekuensi baru untuk sebuah simpul baru yaitu charlist.frek yang mempunyai dua cabang kedua simpul tadi (pinfo1 dan pinfo2). Untuk pinfo1 ditetapkan sebagai cabang kiri dan pinfo2 ditetapkan sebagai cabang kanan. Simpul baru tersebut juga memiliki data berupa jumlah dari nilai piksel yaitu charlist.jml_kode.
6. Simpul baru ini selanjutnya dimasukkan dalam daftar untuk dilakukan pencarian frekuensi terkecil, dan pembuatan simpul baru.
7. Proses ini berlanjut sampai semua simpul telah masuk ke dalam pohon dan frekuensi akar (*root*) dari pohon akan merupakan hasil penjumlahan semua frekuensi yang ada.. Berikut ini listing dari pembentukan pohon Huffman.

```
procedure pohonHuffman;
var
 i,cnt,tmp:integer;
 pinfo1,pinfo2:Phuffinfo;
begin
 pinfo1:=nil;
 pinfo2:=nil;
 cnt:=255;
 while true do
  begin
   tmp:=maxint;
   for i:=0 to cnt do
    begin
     if (charlist[i].frek<tmp) and (charlist[i].frek > 0)
and(charlist[i].ticked=false) then
      begin
       pinfo1:=@charlist[i];
       tmp:=pinfo1.frek;
      end;
    end;
   if pinfo1=nil then
    break;
   pinfo1.ticked:=true;
   tmp:=maxint;
   for i:=0 to cnt do
    begin
     if (charlist[i].frek<tmp) and (charlist[i].frek > 0) and
(charlist[i].ticked=false) then
      begin
       pinfo2:=@charlist[i];
       tmp:=pinfo2.frek;
      end;
```

```
    end;
   if pinfo2=nil then
    break;
   pinfo2.ticked:=true;
   inc(cnt);
   charlist[cnt].frek:=pinfo1.frek+pinfo2.frek;
  charlist[cnt].jml_kode:=pinfo1.jml_kode+pinfo2.jml_kode;
   charlist[cnt].kiri:=pinfo1;
   charlist[cnt].kanan:=pinfo2;
   pinfo1:=nil;
   pinfo2:=nil;
  end;
 rootnode:=@charlist[cnt];
end;
```

3. Pengkodean
   Pemberian kode biner pada pohon Huffman dilakukan dengan memberikan kode pada tiap cabang pada pohon Huffman. Berikut ini langkah-langkah dalam pemberian kode pada pohon Huffman:
   1. Ambil nilai piksel yang akan dikodekan
   2. Dari setiap nilai keabuan yang akan dikodekan telusuri mulai dari akar. Apabila pada cabang kiri terdapat nilai piksel yang sedang ditinjau, maka berikan kode "0". Dan apabila terdapat pada cabang kanan maka berikan simbol "1".
   3. Setiap cabang yang telah ditelusuri, tetapkan sebagai akar.
   4. Ulangi langkah diatas sampai ditemukan daun. Beriukut ini listing dari pemberian kode Huffman.

```
procedure kodeHuffman;
var
 i,j:integer;
 tmpnode:phuffinfo;
 flag:integer;
begin
 for i:=0 to 255 do
  begin
   flag:=-1;
   tmpnode:=rootnode;
   while tmpnode.kiri<>nil do
    begin
     for j:=0 to tmpnode.kiri.jml_kode-1 do
      if tmpnode.kiri.piksel[j]= i then
       begin
        flag:=0;
        tmpnode:=tmpnode.kiri;
        break;
       end;
      if flag=-1 then
       begin
        for j:=0 to tmpnode.kanan.jml_kode-1 do
         if tmpnode.kanan.piksel[j]= i then
```

```
          begin
           flag:=1;
           tmpnode:=tmpnode.kanan;
           break;
          end;
        end;
      if flag=-1 then
       break;
      huffcodes[i].ada:=true;
                huffcodes[i].kode[huffcodes[i].panjangKode]:=flag;
      huffcodes[i].panjangKode:=huffcodes[i].panjangKode+1;
      huffcodes[i].char:=i;
      flag:=-1;
     end;
   end;
end;
```

4. Pembuatan tabel Huffman

   Tabel huffman digunakan untuk menyimpan data-data hasil dari pengkodean huffman. Data-data disini disimpan dalam variabel array. Deklarasi dari variabel array tersebut adalah table: array[0..1495] of byte; Langkah-langkah dalam pembuatan tabel huffman adalah sebagi berikut:

   1. Ambil satu nilai keabuan yang akan disimpan.
   2. Indeks pertama diisi untuk menyimpan nilai keabuan. Misalkan table[0]:=25.
   3. Indeks kedua diisi untuk menyimpan hasil panjang kode huffman dari nilai keabuan yang sedang ditinjau.
   4. Indeks ketiga diisi untuk menyimpan hasil kode huffman. Karena hasil dari panjang kode huffman bervariasi maka semua kode disimpan dalam 8 bit. Berikut ini listing dari pembentukan tabel huffman:

```
procedure TabelHuffman;
var
 i,j,k:integer;
 tmpcode:byte;
 bit:integer;
begin
k:=0;
 bit:=0;
 tmpcode:=0;
 for i:=0 to 255 do
  begin
   if huffcodes[i].ada=false then
    continue;
   table[k]:=i;
   table[k+1]:=huffcodes[i].panjangKode;
   k:=k+2;
   for j:=0 to huffcodes[i].panjangKode-1 do
    begin
     tmpcode:=tmpcode shl 1 or huffcodes[i].kode[j];
     bit:=bit+1;
```

```
 if bit=8 then
  begin
   table[k]:=tmpcode;
   k:=k+1;
   bit:=0;
   tmpcode:=0;
  end;
 end;
if bit>0 then
 begin
  tmpcode:=tmpcode shl (8-bit);
  table[k]:=tmpcode;
  k:=k+1;
  bit:=0;
  tmpcode:=0;
 end;
end;
Jumtabel:=k;
end;
```

5. Penyimpanan hasil kompresi
   Kode hasil proses huffman harus disimpan ke dalam memori komputer . Berikut ini listing dari penyimpanan hasil kompresi.

```
procedure SimpanFile;
var
x,y, i,j,k:integer;
tmpcode:byte;
bit:integer;
bufcount:integer;
PC:PByteArray;
begin
k:=0;
tmpcode:=0;
bit:=0;
h:= fcitra.Image1.Picture.Height;
w:=fcitra.Image1.Picture.Width;
ofile.seek(1,sofrombeginning);
ofile.write(Jumtabel,sizeof(Jumtabel));
ofile.write(table,Jumtabel);
ofile.write(h,sizeof(h));
ofile.write(hh,h);
ofile.write(w,sizeof(w));
ofile.write(ww,w);
ifile.seek(0,soFromBeginning);
for i:=0 to byk do
begin
for j:=0 to huffcodes[inbuffer[i]].panjangKode-1
 do
begin
tmpcode:=(tmpcode shl 1) or huffcodes[inbuffer[i]].kode[j];
```

```
inc(bit);
if bit=8 then
begin
outbuffer[k]:=tmpcode;
inc(k);
bit:=0;
tmpcode:=0;
end;
if k=byk then
begin
ofile.write(outbuffer,byk);
k:=0;
end;
end;
end;
if bit>0 then
begin
tmpcode:=tmpcode shl (8-bit);
outbuffer[k]:=tmpcode;
k:=k+1;
end;
if k>0 then
ofile.write(outbuffer,k);
ofile.seek(0,sofrombeginning);
ofile.write(bit,1);
end;
```

6. Perhitungan Hasil Kompresi
   Untuk mengetahui tingkat efisiensi ukuran file hasil kompresi diketahui dari besarnya nilai rasio kompresi. Semakin besar rasio kompresi maka semakin besar tingkat efisiensi ukuran file hasil kompresi. Rasio kompresi citra dihitung dengan rumus:

$$\text{Rasio kompresi} = 100\% - \left(\frac{\text{ukuran file hasil kompresi}}{\text{ukuran citra semula}} \right) 100\%$$

4. Dekompresi

Langkah- langkah yang dilakukan pada proses dekompresi adalah:

1. Mengembalikan tabel Huffman
2. Mengembalikan pohon Huffman
   Langkah-langkah yang dilakukan untuk mengembalikan pohon Huffman adalah sebagi beriku:
   1. Baca sebuah nilai kebauan
   2. Mulai dari akar untuk setiap bit pada langkah 1, lakukan traversal pada cabang yang bersesuaian.
   4. Ulangi langkah 1, 2 dan 3 sampai bertemu daun. Kodekan rangkaian bit yang telah dibaca dengan karakter di daun.
   5. Ulangi dari langkah 1 sampai semua nilai keabuan habis.
3. Merubah file kompresi dalam bentuk citra
   Setelah kode Huffman dapat dikembalikan ke nilai semula (nilai piksel citra masukan) maka dari nilai-nilai itu diubah kedalam bentuk matrik untuk dilakukan

penggambaran per titik piksel pada *image canvas.* Berikut ini listing dari penggambaran pada *image canvas.*

```
z:=0;
   form1.image1.ClientHeight:=m;
   form1.image1.ClientWidth:=n;
   for i:= 0 to m-1  do
       for j:=0 to n-1   do
         begin
           tmp:=OutBuffer[z];
      form1.image1.Canvas.Pixels[j,i]:=RGB(tmp,tmp,tmp);
           z:=z+1;
         end;
```

5. Contoh kasus

Misalkan sebuah citra *gray scale* yang berukuran 155x153 dengan 8 derajat keabuan seperti gambar di bawah ini. Maka ukuran citra tersebut adalah 155 x 153 x 8 = 189720 bit.

Gambar Contoh Citra

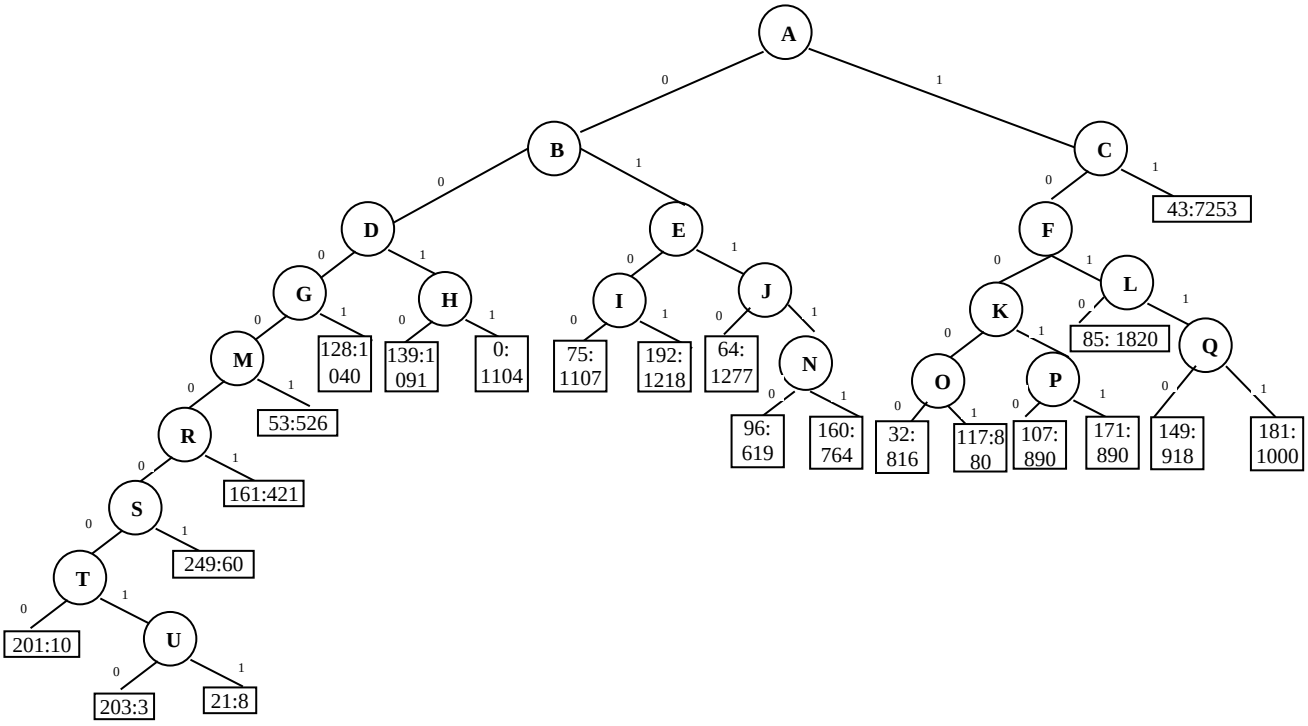
Akan dilakukan kompresi dengan metode Huffman.Berikut ini langkah-langkah untuk melakukan kompresi terhadap citra diatas dengan metode Huffman:

1. Hitung frekuensi dari tiap nilai keabuan untuk setiap baris sampai setinggi citra dan setiap kolom sampai setinggi citra. Tabel 4.1 Tabel Frekuensi Contoh Citra

Keabuan	Frekuensi
0	1104
21	8
32	816
43	7253
53	526
64	1277
75	1107
85	1820
96	619
107	890
117	880
128	1040
139	1091
149	918
160	764
161	421
171	890

181	1000
192	1218
201	10
203	3
249	60

2. Setiap nilai piksel merupakan sebuah simpul yang mempunyai data nilai keabuan dan jumlah frekuensi.
3. Dari simpul-simpul tersebut, cari simpul yang memiliki jumlah frekuensi terkecil pertama. Kemudian cari simpul yang mempunyai jumlah frekuensi terkecil kedua. Dari kedua simpul tadi jumlahkan frekuensinya. Hasil penjumlahan dari kedua simpul tadi menjadi sebuah simpul baru yang memiliki dua cabang kedua simpul tadi. Simpul yang memiliki jumlah frekuensi terkecil pertama menjadi cabang kiri dan simpul yang memiliki jumlah frekuensi terkecil kedua menjadi cabang kanan.
4. Simpul baru ini selanjutnya dimasukkan daftar dalam beserta kedua cabang untuk dilakukan pencarian frekuensi terkecil, dan pembuatan simpul baru.
5. Proses ini berlanjut sampai semua simpul telah masuk ke dalam pohon dan frekuensi akar (*root*) dari pohon akan merupakan hasil penjumlahan semua frekuensi yang ada.
6. Setelah terbentuk pohon Huffman dilakukan penelusuran terhadap pohon Huffman dengan memberikan kode “0” untuk setiap cabaang kiri dan “1” untuk setiap cabang kanan.
7. Maka akan terbentuk pohon Huffman seperti Gambar di bawah ini



Gambar Pohon Huffman

Tabel berikut ini berisi keterangan pada setiap akar dan cabang pada pohon Huffman diatas.

Tabel 4.2 Tabel Keterangan pohon Huffman

Huruf	Simpul	Jumlah Frekuensi
A	201 203 21 249 161 53 128 139 0 75 192 64 96 160 32 117 107 171 85 149 181 43	23715
B	201 203 21 249 161 53 128 139 0 75 192 64 96 160	9248
C	32 117 107 171 85 149 181 43	14467
D	201 203 21 249 161 53 128 139 0	4263
E	75 192 64 96 160	4985
F	32 117 107 171 85 149 181	7214
G	201 203 21 249 161 53 128	2068
H	139 0	2195
I	75 192	2325
J	64 96 160	2660
K	32 117 107 171	3476
L	85 149 181	3738
M	201 203 21 249 161 53	1028
N	96 160	1383
O	32 117	1696
P	107 171	1780
Q	149 181	1918
R	201 203 21 249 161	502
S	201 203 21 249	81
T	201 203 21	21
U	203 21	11

8. Kemudian dilakukan penelusuran pada pohon Huffman. Dimulai dari akar, penelusuran menurun didapatkan simpul tanpa cabang (daun).
9. Begitu didapatkan daun, maka nilai piksel dimasukkan dalam tabel beserta angka-angka bit (“0” atau “1”) dari cabang-cabang yang dilalui untuk sampai pada daun ini.
10. Maka diperoleh kode Huffman dari masing-masing nilai kebuan seperti tabel di bawah ini:

Tabel 4.3 Tabel Kode Huffman

Keabuan	Kode Huffman	Panjang kode
0	0011	4
21	000000011	9
32	10000	5
43	11	2
53	00001	5
64	0110	4
75	0100	4
85	1010	4
96	01110	5
107	10010	5
117	10001	5

128	0001	4
139	0010	4
149	10110	5
160	01111	5
161	000001	6
171	10011	5
181	10111	5
192	0101	4
201	00000000	8
203	000000010	9
249	0000001	7

Kode Huffman dari masing-masing nilai keabuan tersebut disimpan dalam memori komputer. Sebagai contoh dari tabel Huffman diatas akan menyimpan nilai keabuan 0 yang memiliki kode huffnan 0011.

8. Simpan kode-kode Huffman tersebut harus ke dalam memori komputer untuk mempermudah dalam proses dekompresi.
9. Lakukan proses perhitungan hasil kompresi dengan cara mengalikan panjang kode dengan jumlah frekuensi setiap nilai keabuan. Kemudian jumlahkan semua hasil perkalian tersebut. Berikut ini tabel perhitungannya:

Tabel 4.4 Tabel Perhitungan Kompresi

Keabuan	Frekuensi	Panjang kode	Frekuensi * panjang kode
0	1104	4	4416
21	8	9	72
32	816	5	4080
43	7253	2	14506
53	526	5	2630
64	1277	4	5108
75	1107	4	4428
85	1820	4	7280
96	619	5	3095
107	890	5	4450
117	880	5	4400
128	1040	4	4160
139	1091	4	4364
149	918	5	4590
160	764	5	3820
161	421	6	2526
171	890	5	4450
181	1000	5	5000
192	1218	4	4872
201	10	8	80
203	3	9	27
249	60	7	420
		jumlah	88774

Maka ukuran citra setelah pemampatan adalah 88774 bit. Jadi kebutuhan memori telah dapat dikurangi dari 189720 bit menjadi 88774 bit.

Rasio Kompresi adalah $(100\ \% - \frac{88774}{189720} \times 100\%) = 53.21\ \%$

PENUTUP

1. Kesimpulan

1. Teknik kompresi citra dengan metode Huffman pada citra *gray scale* 8 bit dilakukan dengan pembuatan pohon Huffman berdasarkan frekuensi kemunculan nilai piksel. Pohon Huffman akan menghasilkan kode biner yang disimpan dalam tabel Huffman tiap 8 bit.
2. Kode biner yang dihasilkan dari Metode Huffman adalah setiap nilai piksel yang mempunyai frekuensi terbesar mempunyai jumlah bit yang pendek sedangkan nilai piksel yang mempunyai frekuensi kecil mempunyai bit yang lebih panjang..
3. Citra yang mempunyai sebaran nilai piksel tidak merata memiliki rasio kompresi yang relatif besar sedangkan citra dengan nilai piksel yang merata memiliki rasio kompresi yang lebih kecil.
4. Tingkat efisiensi memori file hasil kompresi dengan menggunakan Metode Huffman diukur dari besarnya rasio kompresi yang dihasilkan. Citra yang mempunyai sebaran nilai piksel tidak merata memiliki tingkat efisiensi lebih besar dibandingkan dengan citra dengan nilai piksel yang merata.

2. Saran

1. Untuk dapat lebih melihat dan membuktikan keefektifan, kelebihan dan kelemahan dari algoritma Huffman, perlu diadakannya sebuah penelitian yang bertujuan membandingkan seluruh algoritma kompresi dalam mengompres berbagai citra.
2. Untuk menyempurnakan program kompresi citra dengan metode Huffman pada Borland Delphi 6.0 hendaknya ditambahkan program untuk mengenali file hasil Huffman *coding*.

DAFTAR PUSTAKA

Achmad, Balza, dan Kartika Firdausy. 2005. *Teknik Pengolahan Citra Digital menggunakan Delphi*. Ardi Publising, Yogyakarta.

Fadlisyah, Taufiq, Zulfikar, Fauzan. 2008. *Pengolahan Citra Menggunakan Delphi.* Graha Ilmu, Yogyakarta.

http://mail.informatika.org/~rinaldi/Matdis/2007-2008/Makalah/MakalahIF2153-0708-075.pdf, Tanggal akses: Minggu, 4 Mei 2008 pukul 05:50

http://lecturer.ukdw.ac.id/anton/download/multimedia7.pdf, Tanggal akses: Minggu, 4 Mei 2008 pukul 06:04

http://mail.informatika.org/~rinaldi/Stmik/2005-2006/Makalah2006/MakalahStmik2006-43.pdf, Tanggal akses: Sabtu, 15 Maret 2008 pukul 06:05

http://home.unpar.ac.id/~integral/Volume%209/Integral%209%20No.%201/Perbandingan%20Kinerja%20Algoritma%20Kompresi.pdf, Tanggal akses: Minggu, 4 Mei 2008 pukul 20:01

http://mail.informatika.org/~rinaldi/Matdis/2007-2008/Makalah/MakalahIF2153-0708-010.pdf, Tanggal akses: Minggu, 4 Mei 2008 pukul 20:15

http://mail.informatika.org/~rinaldi/Matdis/2007-2008/Makalah/MakalahIF2153-0708-086.pdf, Tanggal akses: Sabtu, 15 Maret 2008 pukul 06:38

http://en.wikipedia.org/wiki/Data_compression, Tanggal akses: Sabtu, 15 Maret 2008 pukul 06:15

Huffman Coding, http://www.en.wikipedia.org/wiki/Huffman_coding, Tanggal akses : Tanggal akses: Sabtu, 15 Maret 2008 pukul 06:28

Munir, Rinaldi. 2004.Penglahan Citra Digital dengan pendekatan Algoritmik. Informatika, Bandung.

Munir, Rinaldi. 2005. *Algoritma dan Pemrograman dalam bahasa Pascal dan C Edisi ke-3*. Informatika, Bandung.

Practical Huffman Coding, http://www.compressconsult.com/huffman/, Tanggal akses: Minggu, 4 Mei 2008 pukul 05:50

Pranata, Antony. 2002. *Pemrograman Borland Delphi 6.0 Edisi 4.*. Andi Yogyakarta. Yogyakarta.

Santoso, Insap. 1992. *Struktur Data Menggunakan Turbo Pascal 6.0.* Andi Yogyakarta, Yogyakarta.