PENGKODEAN ARITMETIKA UNTUK KOMPRESI DATA TEKS
(*Arithmetic Coding for Text Compression)*

Bib Paruhun Silalahi, Fahren Bukhari, Solikha Nurhudayani[1]

Abstrak

Teknik kompresi merupakan teknik yang dikembangkan dengan tujuan menghasilkan *file* baru yang merupakan hasil pengkodean dari *file* asli dengan ukuran yang lebih kecil. Pengkodean aritmetika merupakan teknik pengkodean statistik karena menggunakan peluang kemunculan suatu karakter pada teks untuk menghasilkan kode kompresi. Penelitian ini mempelajari teknik kompresi teks menggunakan pengkodean aritmetika, serta mengimplementasikannya untuk kasus kompresi dan dekompresi. Hal yang dikaji antara lain: melihat pengaruh variasi karakter di dalam suatu teks terhadap rasio kompresi, waktu kompresi yang dihasilkan serta waktu dekompresinya. Hasil percobaan menunjukkan bahwa rasio kompresi yang tertinggi diperoleh pada saat suatu teks hanya memiliki satu jenis karakter saja. Adapun pengaruh variasi karakter terhadap rasio kompresi adalah semakin banyak variasi karakter yang dimiliki oleh suatu teks maka rasio kompresi akan semakin rendah.

Kata Kunci: Kompresi Data, *Lossless Compression,* Pengkodean Aritmetika, Kompresi Teks, *Arithmetic Coding*

PENDAHULUAN

Latar Belakang

Perkembangan teknologi menyebabkan penggunaan komputer semakin meluas sehingga, sebagian besar data dalam perusahaan atau organisasi sudah disimpan dalam bentuk digital. Data yang besar akan membutuhkan ruang penyimpanan yang besar pada *hard disk.* Selain itu, data berukuran besar yang dikomunikasikan melalui jaringan komputer akan memperbesar waktu pengiriman pada jaringan tersebut.

Untuk mengatasi masalah di atas, dikembangkan suatu teknik kompresi yang bertujuan untuk memperkecil ukuran *file.* Menurut Chrocemore dan Lecroq (2000), penggunaan teknik kompresi data menjadi sangat penting, terutama bila tempat penyimpanan pada *hard disk* terus berkurang bersamaan dengan penambahan data secara teratur pada setiap waktu tertentu. Beberapa teknik kompresi menggunakan peluang kemunculan suatu karakter untuk menghasilkan kode kompresi, misalnya pengkodean Huffman dan pengkodean aritmetika Pengkodean aritmetika diusulkan oleh Elias dan dipresentasikan oleh Abramson dalam bukunya *Information Theory and Coding* pada tahun 1963 (Lelewer dan Hirschberg, 1987). Pengkodean aritmetika mengambil sederetan karakter pada teks dan menggantikannya dengan sebuah bilangan sebagai hasil kompresi (Nelson, 1991). Dengan teknik tersebut, kode unik dihasilkan untuk sederetan karakter tanpa harus membuat kode untuk tiap karakter.

Tujuan

Tujuan penelitian ini adalah:

1. Mempelajari dan memahami teknik kompresi data dengan menggunakan pengkodean aritmetika
2. Menganalisa algoritma pengkodean aritmetika.
3. Mengimplementasikan algoritma tersebut dengan menggunakan bahasa pemrograman Visual Basic, untuk kasus kompresi dan dekompresi.

Ruang Lingkup

Penelitian yang akan dilakukan terbatas pada kompresi data berbentuk teks yang karakternya tergabung dalam 256 kode ASCII.

TINJAUAN PUSTAKA

Kompresi Data

Menurut Lelewer dan Hirschberg (1987), kompresi data sering dihubungkan dengan pengkodean yang merupakan representasi data secara khusus untuk memenuhi suatu kebutuhan. Kompresi teks menghasilkan *file* hasil kompresi

dari suatu *file* teks dengan ukuran yang lebih kecil dari *file* teks sumber.

Dekompresi

Proses dekompresi mengembalikan *file* kompresi menjadi teks awal. Hasil dekompresi tergantung dari sifat kompresi yang digunakan, yaitu *Lossless Compression* atau *Lossy Compression.*

Lossless Compression

Jika telah dilakukan teknik *lossles compression* pada suatu teks, teks yang asli dapat diperoleh kembali secara tepat dari file hasil dekompresi tersebut (Sayood, 2001). Pengkodean aritmetika merupakan teknik kompresi yang bersifat *lossless compression.*

Lossy Compression

Lossy Compression mengakibatkan hilangnya beberapa informasi, dan hasil dekompresi tidak dapat menghasilkan teks yang tepat sama dengan teks asli (Sayood, 2001).

Rasio Kompresi

Rasio Kompresi menunjukkan presentase besarnya kompresi yang dilakukan terhadap *file* asli. Rasio kompresi diperoleh dari persamaan:

$$\text{Rasio kompresi} = \frac{\text{Selisih ukuran}}{\textit{file}\ \text{asli}} \times 100\% \quad (1)$$

Selisih ukuran = *file* asli – *file* kompresi (2)

Semakin tinggi rasio kompresi maka ukuran *file* kompresi yang dihasilkan semakin kecil, yang berarti hasil kompresi semakin bagus.

Pengkodean Aritmetika

Ide dasar pengkodean aritmetika adalah membuat suatu garis peluang dari 0 sampai 1 dan memberi interval pada setiap karakter dari teks *input* berdasarkan peluang kemunculannya. Semakin tinggi peluang yang dimiliki suatu karakter maka semakin besar interval yang akan diperoleh (Campos, 1999). Setelah semua karakter memiliki interval maka dilakukan pengkodean untuk menghasilkan satu bilangan *output*. Sebagai contoh, untuk teks ABCAD, distribusi peluang dari teks tersebut dapat dilihat pada tabel berikut:

Tabel 1. Distribusi peluang untuk teks ABCAD

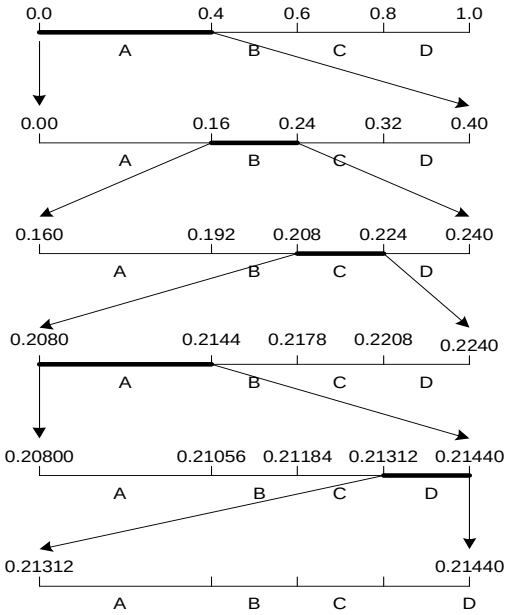
KARAKTER	PELUANG
A	0.4
B	0.2
C	0.2
D	0.2

Tabel 2 di bawah ini menunjukkan interval yang diberikan pada masing-masing karakter:

Tabel 2. Rentang karakter pada garis peluang

KARAKTER	PELUANG	INTERVAL
A	0,4	[0, 0.4)
B	0,2	[0.4 , 0.6)
C	0,2	[0.6 , 0.8)
D	0,2	[0.8 , 1.0)

Terbacanya karakter pertama pada proses pengkodean akan mempersempit panjang garis peluang menjadi sebesar interval yang dimiliki oleh karakter tersebut. Pada garis peluang yang baru akan dilakukan partisi lagi dengan porsi yang sama seperti pada garis peluang sebelumnya. Ilustrasi dari keseluruhan proses partisi dari garis peluang untuk teks ABCAD dapat dilihat pada Gambar 1 di bawah ini.



Gambar 1. Proses Partisi pada garis peluang untuk teks ABCAD

Sebagai hasil kompresi diambil satu nilai yang berada di dalam interval terakhir, misalnya nilai batas bawahnya yaitu 0.21312.

Algoritma pengkodean aritmetika dapat menghitung garis peluang yang baru tanpa harus melakukan pembagian interval dari setiap garis peluang yang baru. Dasar dari algoritma pengkodean aritmetika adalah sebagai berikut:

AR-Encode

1 Cari peluang dari tiap karakter pada teks yang akan dikodekan
2 $l_{(0)} \leftarrow 0$, $h_{(0)} \leftarrow 1$, $i \leftarrow 1$
3 while not EOF (*fin)*, baca karakter ke *i*
4 do $l_{(i)} \leftarrow l_{(i-1)} + (h_{(i-1)} - l_{(i-1)}) * l_{\text{interval (i)}}$
5 $h_{(i)} \leftarrow l_{(i-1)} + (h_{(i-1)} - l_{(i-1)}) * h_{\text{interval (i)}}$
6 $i \leftarrow i + 1$
7 return $l_{(i)}$

fin : *file input*
$l_{(i)}$: batas bawah interval ke-*i*
$h_{(i)}$: batas atas interval ke-*i*
$l_{\text{interval (i)}}$: batas bawah interval dari karakter ke-*i*
$h_{\text{interval (i)}}$: batas atas interval dari karakter ke-*i*

Untuk input teks ABCAD hasil yang diperoleh melalui algoritma di atas dapat dilihat pada Tabel 3.

Tabel 3. Proses pengkodean dengan menggunakan algoritma

Karakter	$h_{(i-i)} - l_{(i-i)}$	*l*	*h*
		0	1
A	1	0	0.4
B	0.4	0.16	0.24
C	0.08	0.208	0.224
A	0.016	0.208	0.2144
D	0.0064	0.21312	0.2144

Untuk proses pendekodean nilai yang dibutuhkan untuk adalah nilai *output (V)* yang dihasilkan pada proses pengkodean. Pertama dilakukan pencarian terdapat pada interval manakah nilai tersebut sehingga diperoleh karakter yang memiliki interval tersebut. Selanjutnya nilai *V* di-*update* berdasarkan persamaan:

$$V_{(i)} \leftarrow \frac{V_{(i-1)} - l_{\text{interval (i-1)}}}{h_{\text{interval (i-1)}} - l_{\text{interval (i-1)}}} \qquad (3)$$

Proses tersebut akan berlanjut sampai $V_{(i)}$ mencapai nilai 0. Berikut algoritma untuk proses pendekodan:

AR-Decode

1 $i \leftarrow 0$, $V_{(i)} \leftarrow$ *Nilai* hasil kompresi
2 while $V_{(i)}$ *!*= 0
3 do cari a_i di mana $V_{(i)} \in I_{(ai)}$
5 write a_i ke dalam *fout*
6 $V_{(i)} \leftarrow \frac{V_{(i-1)} - l_{\text{interval (i-1)}}}{h_{\text{interval (i-1)}} - l_{\text{interval (i-1)}}}$
7 $i \leftarrow i + 1$

fout : hasil dekode
$I_{(ai)}$: interval karakter ke- *i*
$V_{(i)}$: Nilai *Output* hasil kompresi
$l_{(i)}$: batas bawah interval ke-*i*
$l_{\text{interval (i)}}$: batas bawah interval dari karakter ke-*i*
$h_{\text{interval (i)}}$: batas atas interval dari karakter ke-*i*

Untuk kasus *file* teks ABCAD di atas, nilai *l* yang diperoleh adalah 0.21312, dan nilai interval pada masing-masing karakter adalah $I_{(A)} = [0, 0.4)$, $I_{(B)} = [0.4, 0.6)$, $I_{(C)} = [0.6, 0.8)$, $I_{(D)} = [0.8, 1.0)$. 0.21312 berada pada interval 0 dan 0.4, sehingga diketahui karakter pertama adalah A. Berdasarkan persamaan (3), nilai *V* akan dikurangi dengan batas bawah dari interval yang diperoleh, yaitu 0 lalu dibagi dengan hasil pengurangan antara 0.4 dan 0. Sehingga nilai $V = (0.21312 - 0) / (0.4 - 0) = 0.5328$. Nilai 0.5328 tersebut berada dalam interval [0.4, 0.6), sehingga karakter kedua dari teks adalah B dan nilai *V* menjadi 0.664, begitu seterusnya hingga nilai *V* mencapai 0.

METODE PENELITIAN

Pengumpulan data

Data yang dikumpulkan adalah data yang akan digunakan untuk pengujian progam. Data tersebut berupa *file* teks (*.txt) yang berukuran tidak lebih dari 100000 *bytes*.

Perencanaan Progam

Program yang dibuat merupakan program aplikasi dari pengkodean aritmetika untuk kompresi dan dekompresi. Ketika *file input* dimasukkan ke dalam program, setiap jenis karakter yang terdapat pada *file* akan dihitung jumlahnya. Selanjutnya program akan melakukan pengkodean berdasarkan hasil perhitungan tersebut.

Analisis

Pada tahap ini dilakukan identifikasi kebutuhan informasi yang akan ditampilkan, diantaranya rasio kompresi, waktu kompresi dan dekompresi, serta frekuensi dari setiap jenis karakter dari *file input* baik sebelum maupun setelah dilakukan kompresi dan dekompresi dengan disertai tampilan grafiknya. Selain itu juga dilakukan analisis kompleksitas dari program pengkodean aritmetika.

Desain Program

Untuk desain masukan pada program kompresi, *input* data berupa *file* teks (*.txt) dan desain keluaran berupa *file* hasil kompresi yaitu *file* dengan format *.arit. Untuk desain masukan pada program dekompresi, *input* berupa *file* dengan format *.arit dan untuk desain keluaran, berupa *file* teks.

Implementasi

Implementasi pengkodean Aritmetika dilakukan pada *hardware* dan *software* dengan spesifikasi sebagai berikut:

- *Hardware*
  - Processor 850 Mhz
  - RAM 128 Mb
- *Software*
  - Visual Basic 6
  - OS Windows XP

Proses penempatan pada interval. Pada tahap ini akan dihitung frekuensi karakter dan total karakter yang ada pada teks *input*. Sebelumnya, seluruh karakter yang terdapat pada teks *input* disimpan ke dalam satu *array* dan direpresentasikan dalam kode ASCII (bertipe *byte*). Setelah itu dilakukan pemetaan batas bawah dan batas atas dari tiap jenis karakter ke dalam interval.

Untuk menyimpan hasil perhitungan yang diperoleh di atas, digunakan suatu *record* yang berisi informasi yang dibutuhkan untuk pengkodean, yaitu frekuensi, nilai batas bawah dan nilai batas atas dari tiap jenis karakter. Pendeklarasian *record* tersebut dapat dilihat pada potongan program di bawah ini:

```
Private Type DataChar
    Count As Long
    Low As Variant
    High As Variant
End Type
Char(256) As DataChar
```

Informasi jenis karakter dan frekuensinya akan dimasukkan ke dalam suatu *array* bertipe *byte* bersama dengan hasil pengkodean. *Array* tersebut kemudian akan disimpan di dalam *file* hasil kompresi. Informasi karakter dan frekuensinya perlu disimpan untuk dijadikan acuan pada saat melakukan dekompresi.

Teknik penyimpanan frekuensi dari karakter. Saat akan menyimpan frekuensi dari suatu jenis karakter terdapat suatu masalah karena terbatasnya nilai maksimum yang dapat disimpan pada *array*. *Array* yang digunakan bertipe byte dengan nilai maksimum 255. Untuk karakter yang memiliki frekuensi lebih dari 255 buah dibutuhkan suatu teknik untuk mengatasi permasalahan di atas. Teknik tersebut terdiri dari dua tahap, yaitu:

1. $Array(n) = \text{Int}(\ Frek(i) / 256\ )$
2. $Array(n + 1) = Frek(i)$ And 255

dengan n merupakan nilai indeks dari *array* dan $Frek(i)$ merupakan frekuensi dari karakter ke-i.

Tahap pertama melakukan pembagian frekuensi dari karakter terhadap nilai 256 yang merupakan banyaknya karakter ASCII. Hasil pembagian tersebut menunjukkan nilai kelipatan dari frekuensi suatu karakter terhadap 256. Tahap kedua melakukan operasi AND antara representasi biner dari frekuensi suatu karakter dengan 11111111 yang dalam bilangan desimal berarti 255.

Berdasarkan tahap di atas, frekuensi dari setiap karakter akan disimpan menjadi dua indeks *array*. *Array* pada tahap satu akan selalu bernilai 0 jika frekuensi dari karakter kurang dari 255. Sebagai contoh, akan dilihat bagaimana penyimpanan frekuensi karakter di atas 255, yaitu 500.

Tahap 1 : $\text{Int}(500 / 256) = 1 \rightarrow \text{Array}(1)$

Tahap 2 : $111110100 \rightarrow 500$

$11111111 \rightarrow 255$

——————— AND

$11110100 \rightarrow 244 \rightarrow \text{Array}(2)$

Karakter dengan frekuensi antara 256 sampai dengan 65535 nilainya akan dipecah ke dalam dua indeks *array*.

Teknik Pengkodean. Masalah yang dihadapi saat implementasi adalah garis peluang yang semakin menyempit dan tidak adanya nilai *output* yang dihasilkan sampai seluruh karakter terbaca. Ketika jumlah karakter pada suatu *file* bertambah besar maka besar interval antara batas bawah (l) dan batas atas (h) suatu karakter akan semakin sempit. Untuk memperoleh interval yang dapat merepresentasikan karakter secara unik, diperlukan ketelitian yang tinggi bersamaan dengan meningkatnya jumlah karakter pada *file* teks (Sayood, 2000). Untuk mengatasi situasi tersebut harus dilakukan penskalaan ulang terhadap suatu interval.

Hal pertama yang harus dilakukan adalah menentukan panjang bit untuk representasi biner dari l dan h. Dengan panjang bit n, interval [0 , 1) diperluas menjadi $[0, 2^n–1)$. Untuk memutuskan

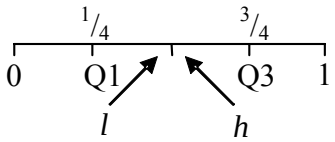
panjang bit yang akan digunakan, harus dipastikan bahwa panjang interval yang dihasilkan dapat menyediakan jarak yang cukup untuk merepresentasikan perbedaan terkecil antara nilai *l* dan *h* sehingga nilai *l* dan *h* akan tetap memiliki nilai yang berbeda pada saat interval semakin menyempit. Setelah diperoleh interval dari tiap jenis karakter maka pengkodean dapat dilakukan. Proses *update* nilai *l* dan *h* pada saat setiap karakter terbaca akan dilakukan berdasarkan persamaan berikut:

$$l \leftarrow l + \frac{(h - l + 1) \times \text{batas bawah karakter ke-}i}{\text{Total Karakter}} \quad (4)$$

$$h \leftarrow l + \frac{(h - l + 1) \times \text{batas atas karakter ke-}i}{\text{Total Karakter}} - 1 \quad (5)$$

Setelah memperoleh nilai *l* dan *h* dari persamaan (4) dan (5), nilai *output* dapat diperoleh dengan melakukan pengulangan berdasarkan kondisi-kondisi di bawah, seperti yang dituliskan oleh Howard dan Vitter (1992):

1. Jika *l* dan *h* tidak berada dalam interval [0, Nilai tengah), [Kuartil pertama, Kuartil ketiga) atau [Nilai tengah, 2^n–1), maka hentikan pengulangan dan baca karakter selanjutnya dari teks *input.*
2. Jika *l* dan *h* berada di bawah nilai tengah interval [0, 2^n–1), maka *output*-nya adalah bit 0 dan memperluas interval. Bit 0 menggambarkan bahwa ketika kedua nilai *l* dan *h* sama-sama berada di bawah nilai tengah, maka representasi biner *l* dan *h* akan memiliki bit terkiri (*Most Significant Bit*) yang sama, yaitu 0.
3. Jika nilai *l* dan *h* berada di atas nilai tengah interval [0, 2^n–1), maka maka *output*-nya adalah bit 1 dan memperluas interval. Bit 1 menggambarkan bahwa ketika *l* dan *h* sama-sama berada di atas nilai tengah dari interval [0, 2^n–1), maka bit terkiri dari representasi biner *l* dan *h* akan sama-sama bernilai 1.
4. Jika *l* dan *h* berada di antara kuartil pertama dan kuartil ketiga dari interval [0, 2^n–1), maka terjadi *underflow.* Kondisi ini terjadi karena *l* dan *h* semakin dekat tapi tidak memiliki bit terkiri yang sama, misalnya *l* = 011111110101 dan *h* = 100000001100. Gambar 2 di bawah ini menggambarkan kondisi *underflow.*



Gambar 2. Kondisi *underflow* pada garis peluang.

Untuk kondisi seperti di atas *l* harus pindah ke atas nilai tengah atau *h* pindah ke bawah nilai tengah. Jika *h* yang pindah maka bit yang akan dikeluarkan 01111111, dan jika *l* yang pindah maka bit yang dikeluarkan adalah 10000000. Jadi dapat dilihat bit apapun yang akan dikeluarkan (0 atau 1), akan diikuti dengan *n* bit yang berlawanan. Solusi untuk kondisi tersebut adalah mengalikan *l* dan *h* dengan 2, dan suatu variabel misalnya *counter* akan di-*increment* untuk menghitung terjadinya perkalian tersebut dan tidak mengeluarkan bit apapun. Ketika akhirnya interval memenuhi salah satu kondisi 2 atau 3, maka bit 0 atau 1 akan dikeluarkan dengan diikuti *n* bit yang berlawanan.

Hasil pengkodean dari teks input merupakan serangkaian bit yang dikeluarkan selama proses pengkodean di atas. Berikut algoritma kompresi pada pengkodean aritmetika:

AR-Compression (*i*)

Baca karakter ke-i dari teks input

$$l \leftarrow l + \frac{(h - l + 1) \times \text{BatasBawah karakter ke-}i}{\text{Total Karakter}}$$

$$h \leftarrow l + \frac{(h - l + 1) \times \text{BatasAtas karakter ke-}i}{\text{Total Karakter}} - 1$$

```
Do
  If h < NilaiTengah Then
        InsertBit(0)
        l ← 2 * l
        h ← 2 * h + 1
        While (Counter > 0)
              InsertBit(1 – 0 ,1)
              Counter ← Counter – 1
  Else if l > NilaiTengah Then
        InsertBit(1)
        l ← 2 * (l – NilaiTengah)
        h ← 2 * (h – NilaiTengah) + 1
        While (Counter > 0)
              InsertBit(1 – 1 ,1)
              Counter = Counter – 1
  Else if (l >= KuartilPertama dan h <=KuartilKetiga) _
  Then
        l ← 2 * (l – KuartilPertama)
        h ← 2 * (h – KuartilPertama) + 1
        ‘menghitung kondisi underflow
```

```
            Counter ← Counter  + 1
    Else
            Exit Do
Loop
```

Penyimpanan kode biner hasil kompresi. Kode hasil kompresi tidak disimpan dalam bentuk biner ke dalam *file* kompresi, tetapi dikonversi terlebih dahulu ke dalam nilai ASCII, kemudian baru disimpan ke dalam *file* kompresi. Konversi dilakukan setiap 8 bit, karena satu kode ASCII akan membutuhkan tempat sebesar 1 *byte* yang terdiri dari 8 bit. Setiap satu bit yang masuk akan dikonversi menjadi bilangan desimal berdasarkan persamaan berikut:

$$\text{NilaiKode} = \text{Nilai Kode} + \text{bit} * 2^n \qquad (6)$$

dengan *n* merupakan posisi dari masing-masing bit dalam kode biner tersebut. Perhitungan dimulai dari *n* bernilai 7 untuk bit terkiri, sehingga ketika mencapai bit ke-8 maka *n* akan bernilai 0. Untuk bit ke-9, *n* akan kembali bernilai 7. Proses akan terus berulang seperti itu sampai bit hasil kompresi habis. Proses konversi tidak menunggu sampai semua bit hasil kompresi diperoleh, melainkan bersamaan dengan setiap satu bit dihasilkan. Di bawah ini adalah algoritma untuk mengkonversi kode biner hasil kompresi menjadi kode ASCII.

```
InsertBit(Bit , n)
For i = n – 1  To 0 Step -1
'pada awal program Power sudah diset bernilai 7
   fout ← fout + (Bit * 2 ^ Power)
   Power ← Power - 1
   BitCount ← BitCount + 1
   If BitCount = 8 Then
      'memasukkan nilai bit ke dalam file output
      ArrayToSave(IndexArray) ← fout
      fout ← 0
      BitCount ← 0
      IndexArray ← IndexArray + 1
      Power ← 7
   End If
Next
```

Proses perhitungan frekuensi saat dekompresi. Proses ini diperlukan untuk menghitung jenis karakter beserta frekuensinya. Pada saat kompresi, frekuensi suatu karakter disimpan dalam dua indeks *array* melalui dua tahap. Nilai-nilai pada *array* tersebut diproses lagi ke dalam dua tahap pada [0.4, 0.6)untuk memperoleh nilai frekuensi, yaitu:

1. $Frek(i) = Array(n)$
2. $Frek(i) = Freq(i) * 256 + Array(n+1)$

dengan *n* adalah indeks dari *array* dan $Frek(i)$ adalah frekuensi untuk karakter ke-*i*. Sebagai contoh, pada tahap kompresi telah dihitung bagaimana penyimpanan frekuensi ke dalam *array* untuk karakter dengan frekuensi 500. Sekarang akan kita lihat bagaimana frekuensi diperoleh kembali berdasarkan nilai *array* yang telah disimpan pada saat kompresi.

$Array(1) = 1$
$Array(2) = 244$
Tahap 1 : $Frek(i) = 1$
Tahap 2 : $Frek(i) = 1*256 + 244 = 500$
→ Diperoleh nilai frekuensi = 500

Tahap di atas menghitung kembali frekuensi yang telah disimpan ke dalam dua indeks *array* pada saat kompresi.

Teknik Pendekodean. Proses dekompresi atau dekode merupakan operasi kebalikan dari proses pengkodean yang telah dijelaskan di atas. Pertama, diambil 8 bit terdepan dari bit hasil kompresi dan dimasukkan ke dalam suatu variabel, misalnya *tag*. Lalu diberikan nilai awal pada interval dengan $l = 0$ dan $h = 2^n – 1$. Nilai *tag* lalu akan dimasukkan ke dalam persamaan

$$value \leftarrow \frac{(tag - l + 1)*\text{Total Karakter} - 1}{h - l + 1} \qquad (7)$$

Karakter hasil pendekodean merupakan karakter dengan interval yang mengandung nilai *value* tersebut. Proses selanjutnya tidak jauh berbeda dengan proses pengkodean. Nilai *tag* di-*update* dengan mengeluarkan bit terdepannya dan memasukkan bit hasil kompresi selanjutnya ke bagian belakang *tag*. Dalam bilangan desimal, proses tersebut sama dengan mengalikan nilai *tag* dengan dua lalu ditambah dengan bit selanjutnya dari kode hasil kompresi. Berikut algoritma dekompresi pada pengkodean aritmetika.

AR-Decompression (*IndexToSave)*

$$value \leftarrow \frac{(tag - l + 1)*\text{ Total Karakter} - 1}{h - l + 1}$$

```
for i = 1 until Total jenis karakter
   'mencari interval yang tepat untuk memperoleh
   karakter
   if ( Batas bawah karakter ke-i <= value <   Batas atas
   karakter ke-i ) Then
      FileSave(IndexToSave) = karakter ke-i
      Else
         i ← i + 1
Next
```

$$l \leftarrow l + \frac{(h - l + 1) \times \text{BatasBawah karakter ke-}i}{\text{TotalKarakter}}$$

$$h \leftarrow l + \frac{(h - l + 1) \times \text{BatasAtas karakter ke-}i}{\text{TotalKarakter}} - 1$$

Do
  If $h <$ Nilai_Tengah **then**
    $l \leftarrow l * 2$
    $h \leftarrow h * 2 + 1$
    $t \leftarrow t * 2 +$ bit selanjutnya dari input
  Else if $l >=$ Nilai_Tengah **then**
    $l \leftarrow 2 * (l$ - Half)
    $h \leftarrow 2 * (h$ - Half) + 1
    $t \leftarrow 2 * (t -$ Half) + bit selanjutnya dari input
  Else if $l >=$ Kuartil Pertama dan $h <=$Kuartil_Ketiga **Then**
    $l \leftarrow 2 * (l -$ Kuartil Pertama)
    $h \leftarrow 2 * (h -$ Kuartil Pertama) + 1
    $t \leftarrow 2 * (t -$ Kuartil Pertama) + bit selanjutnya dari input
  Else
    Exit Do
Loop

Pengujian Program

Pengujian dilakukan untuk mengetahui apakah *file* hasil dekrompesi dapat kembali seperti semula dan memiliki ukuran yang sama seperti *file* awal. Pengujian dilakukan terhadap *file* teks dengan rentang ukuran:

- Kurang dari 20000 *bytes*
- Antara 20000 *bytes* sampai 50000 *bytes*
- Di atas 50000 *bytes* dan kurang dari 100000 *bytes*

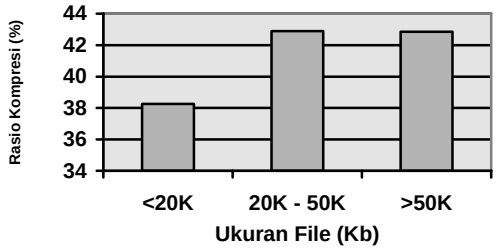
HASIL DAN PEMBAHASAN

Penambahan karakter pada *file* kompresi

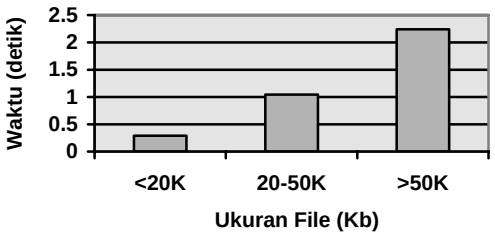
Pada bagian depan *file* kompresi juga disimpan infomasi mengenai banyaknya jenis karakter, kode ASCII karakter dan frekuensi karakter. Satu *byte* pertama pada *file* kompresi digunakan untuk menyimpan informasi banyaknya jenis karakter. Selanjutnya, setiap satu karakter akan memakan tempat sebesar tiga *byte* dalam *file* kompresi. Satu *byte* pertama untuk menyimpan kode ASCII dari karakter, dan dua *byte* sisanya untuk menyimpan frekuensi.

Hasil kompresi pengkodean Aritmetika

Pengujian dilakukan sebanyak lima kali ulangan terhadap masing-masing *file* dalam tiga rentang ukuran di atas. Gambar 3 menunjukkan rata-rata rasio kompresi dan Gambar 4 menunjukkan rata-rata waktu kompresi.



Gambar 3. Grafik rata-rata rasio kompresi



Gambar 4. Grafik rata-rata waktu kompresi

Pada beberapa kasus, dapat terjadi pembengkakan ukuran *file* setelah dikompresi, hal ini disebabkan karena pengaruh dari banyaknya variasi karakter dan adanya penambahan karakter yang telah disebutkan di atas. Untuk melihat berbagai kemungkinan hasil kompresi, selain terhadap tiga rentang *file* di atas, pengujian juga dilakukan terhadap *file* dengan kasus khusus, yaitu:

***1. File* yang memiliki hanya satu jenis karakter.** Pengujian dilakukan terhadap *file* berukuran 280 *bytes* yang hanya memiliki karakter a, *file* kompresi berukuran 8 *bytes* dengan rasio kompresi 97,14%. Struktur dari *file* kompresi ini berarti empat *bytes* untuk informasi mengenai karakter dan empat *bytes* sisanya merupakan hasil pengkodean. Program juga diuji terhadap *file* berukuran 5548 *bytes* dan menghasilkan *file* kompresi berukuran 8 *bytes* dengan rasio kompresi 99,86%. Dapat dilihat jika teks *input* hanya memiliki satu jenis karakter maka selalu menghasilkan *file* kompresi sebesar 8 *bytes*, karena nilai l dan nilai h tidak berubah sepanjang proses pengkodean sehingga tidak ada bit yang dikeluarkan. Pada tahap akhir, bit yang dihasilkan hanyalah bit representasi biner dari l untuk mengakhiri proses pengkodean. Semakin besar ukuran *file* sumber maka semakin tinggi rasio kompresi yang dihasilkan.

***2. File* dengan satu atau beberapa karakter yang memiliki nilai peluang mendekati satu.** Pengujian dilakukan terhadap *file* berukuran

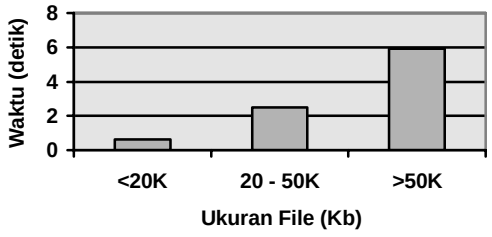
65212 *bytes* dengan frekuensi karakter a sebanyak 53593 dan menghasilkan *file* kompresi sebesar 12591 *bytes* dengan rasio kompresi 80,69%. Dari hasil di atas dapat diketahui jika teks *input* memiliki karakter yang peluangnya mendekati satu, maka rasio kompresi akan tinggi.

***3. File* dengan karakter yang memiliki frekuensi yang sama.** Pengujian dilakukan terhadap *file* berukuran 280 *bytes* yang terdiri dari karakter abcde. *File* kompresi berukuran 101 *bytes* dengan rasio kompresi 63,93%. Pengujian juga dilakukan terhadap *file* dengan ukuran yang sama namun terdiri dari karakter abcdefghij. *File* kompresi berukuran 151 *bytes* dengan rasio kompresi 46,07%. Rasio kompresi yang dihasilkan oleh *file* yang kedua lebih kecil karena variasi karakter *file* yang kedua lebih banyak sehingga penambahan *byte* pada *file* kedua lebih banyak daripada *file* yang pertama.

***4. File* yang memiliki 256 karakter ASCII.** Pengujian dilakukan terhadap *file* berukuran 256 *bytes* yang memiliki semua 256 karakter ASCII dengan frekuensi masing-masing satu. *File* kompresi dari *file* tersebut berukuran 1029 *bytes*, dan rasio kompresi -301,95%. Kemudian diuji lagi dengan ukuran *file* yang lebih besar yaitu 55296 *bytes*, menghasilkan *file* kompresi berukuran 56012 *bytes* dengan rasio kompresi -1,29%. Dari hasil tersebut dapat dilihat bahwa semakin banyak ragam karakter yang dimiliki oleh teks *input* maka rasio kompresi akan semakin rendah.

Hasil dekompresi pengkodean Aritmetika

Dekompresi dijalankan terhadap *file* hasil kompresi di atas. Gambar 5 menunjukkan rata-rata waktu dekompresi.



Gambar 5. Diagram rata-rata waktu dekompresi

Waktu dekompresi lebih lama dibanding waktu kompresi karena ada pencarian interval dari awal garis peluang untuk memperoleh karakter yang akan didekodekan. Semakin besar nilai kode ASCII dari karakter yang akan didekodekan maka waktu pencarian akan semakin lama.

Analisis program pengkodean Aritmetika

Program pengkodean aritmetika secara garis besar terbagi menjadi dua, yaitu kompresi dan dekompresi. Proses kompresi memiliki dua tahap:

1. Pembagian interval dan penyimpanan frekuensi setiap jenis karakter. Untuk melakukan pembagian interval, dilakukan perhitungan frekuensi karakter lalu disimpan ke dalam *array* dan penentuan batas bawah dan batas atas dari setiap jenis karakter. Berikut algoritma untuk keseluruhan proses tersebut:

```
For karakter pertama To karakter terakhir
   Hitung frek masing-masing karakter
Next
Index← 1
For x = 0 To 255
   If Frek(karakter) > 0 Then
      VariasiKarakter ← VariasiKarakter + 1
      ArrayToSave(Index) ← ASCII(karakter)
      Index← Index + 1
      ArrayToSave(Index) ←Int(Frek(karakter) / 256)
      Index← Index + 1
      ArrayToSave(Index) ← Frek(karakter) AND 255
      Index ← Index + 1
      BatasBawah(karakter) ← TotalKarakter
       TotalKarakter ← TotalKarakter + Frek(karakter)
      BatasAtas(karakter) ← TotalKarakter
   End If
Next
```

Untuk proses perhitungan frekuensi tiap jenis karakter akan dilakukan sebanyak *n* kali, dengan *n* sebagai jumlah karakter dalam teks *input*, sehingga kompleksitas proses tersebut adalah sebanyak *O(n).* Sedangkan untuk proses penyimpanan frekuensi dan penempatan karakter ke dalam interval akan dilakukan maksimum sebanyak 255 kali. Secara keseluruhan tahap ini memiliki kompleksitas sebanyak *O(n).*

2. Proses pengkodean. Proses pengkodean dapat dilihat pada algoritma di bawah ini:

While not EOF
1 Baca karakter ke-i dari teks input

2 $$l \leftarrow l + \frac{(h - l + 1) \times \text{BatasBawah karakter ke-}i}{\text{Total Karakter}}$$

3 $$h \leftarrow l + \frac{(h - l + 1) \times \text{BatasAtas karakter ke-}i}{\text{Total Karakter}} - 1$$

```
4   Do
5       If h < NilaiTengah Then
6           InsertBit(0)
7           l ← 2 * l
8           h ← 2 * h + 1
```

```
9               While (Counter > 0)
10                  InsertBit(1 – 0 ,1)
11                  Counter = Counter – 1
12          Else if l > NilaiTengah Then
13              InsertBit(1)
14              l ← 2 * (l – NilaiTengah)
15              h ← 2 * (h – NilaiTengah) + 1
16              While (Counter > 0)
17                  InsertBit(1 – 1 ,1)
18                  Counter ← Counter – 1
19          Else if (l >= KuartilPertama dan
            h <=KuartilKetiga) Then
20              l ← 2 * (l – KuartilPertama)
21              h ← 2 * (h – KuartilPertama) + 1
                'menghitung kondisi underflow
22              Counter ← Counter  + 1
23          Else
24              Exit Do
        Loop
25      i ← i + 1
```

Baris 1, 2 dan 3 pada algoritma di atas dilakukan sebanyak *n*, dengan *n* adalah jumlah karakter pada teks input. Selanjutnya untuk proses pengulangan akan dilakukan maksimum sebanyak panjang bit yang digunakan. Pada penelitian ini, interval yang digunakan adalah [0, 2^{30}–1), sehingga panjang bit adalah 30. Baris 5 sama dengan membandingkan bit terkiri *(MSB)* dari representasi biner *l* dan *h*. Jika *MSB l* dan *h* sama-sama bernilai 0, maka *h* < Nilai tengah.

Sebagai contoh:

l = 17500000, *h* = 2500000

- Representasi biner *l* dan *h*:
  l = 000000000101010110011111110000
  h = 000000001001100010010110100000
- *MSB* dari *l* dan *h* sama-sama bernilai 0. Selanjutnya *MSB* tersebut akan dikeluarkan (digeser ke kiri sebanyak 1 bit) dari nilai *l* dan *h*, lalu menambahkan bit 0 di belakang *l* dan bit 1 ke belakang *h* menjadi:
  l = 000000000101010110011111100000
  h = 000000010011000100101101000001
  Proses di atas (pergeseran dan penambahan bit) sama saja dengan melakukan baris 7 dan 8 pada algoritma pengkodean di atas.

Proses pembandingan *l* >= NilaiTengah pada baris 12 juga sama dengan membandingkan *MSB* dari *l* dan *h*. Jika *MSB* dari *l* dan *h* sama-sama bernilai 1, maka *l* >= NilaiTengah. Selanjutnya pergeseran dan penambahan bit juga dilakukan pada representasi biner *l* dan *h*. Pergeseran dan penambahan bit tersebut sama dengan melakukan baris ke 14 dan 15 pada algoritma pengkodean di atas. Proses pembandingan pada baris 19 untuk kasus jika *MSB* dari *l* dan *h* tidak sama namun bit kedua dari kiri (*MSB* kedua) *l* bernilai 1 dan bit kedua dari kiri *h* bernilai 0. Sebagai contoh:

l = 0**1**11111000000000000000000000000
h = 1**0**11010000011111111111111111111

Pada kasus di atas, *MSB* kedua dari *l* dan *h* dibalik nilainya kemudian dilakukan pergeseran ke kiri sebanyak 1 bit lalu menambahkan bit 0 ke belakang *l* dan bit 1 ke belakang *h* menjadi:

l = **0**11111000000000000000000000000
h = **1**11010000011111111111111111111

Proses pergeseran di atas sama saja dengan melakukan baris 20 dan 21 pada algoritma pengkodean. Berdasarkan proses yang telah dijelaskan di atas, baris 5 sampai 24 pada algoritma pengkodaan akan dilakukan maksimum sebanyak 30 kali karena pergeseran bit hanya dapat dilakukan sampai bit ke 30. Setelah bit ke 30 digeser ke kiri maka nilai *l* dan *h* akan menjadi:

l = 00000000000000000000000000000
h = 11111111111111111111111111111

Nilai *l* dan *h* di atas sudah tidak lagi memenuhi kondisi pada baris 5, 12 dan 19, sehingga proses akan keluar dari pengulangan dan membaca karakter selanjutnya dari teks *input.* Berdasarkan proses di atas, baris 1, 2 dan 3 akan dilakukan sebanyak *n*, sedangkan baris 5 sampai 24 akan dilakukan maksimum sebanyak 30 kali, sehingga kompleksitas algoritma pengkodean adalah *O(n).*

Proses dekompresi juga memiliki dua tahap di dalamnya, yaitu:

1. Proses penghitungan frekuensi dan penempatan interval. Frekuensi yang telah disimpan ke *file* kompresi akan dibaca satu persatu dan dihitung. Setelah frekuensi diperoleh, dilakukan proses penempatan ke dalam *array*. Berikut algoritma yang digunakan untuk proses penghitungan frekuensi dan penempatan interval:

```
For i = 1 To TotalJenisKarakter
   ReadChar ← Masukkan karakter yang terbaca
   CharList(i) ← ReadChar
   index ← index + 1
   Frek(karakter) ← Array(index)
   index ← index + 1
   Frek(karakter) ← Frek(karakter) * 256 + Array(index)
   index ← index + 1
   Frek(karakter) ← Frek(karakter) * 256 + Array(index)
   index ← index + 1
   Freq(i) ← Frek(karakter)
   cum_freq(i) ← cum_freq(i - 1) + Freq(i)
   BatasBawahKarakter ← TotalKarakter
   TotalKarakter ← TotalKarakter + Frek(karakter)
   BatasAtasKarakter ← TotalKarakter
Next
```

Keseluruhan proses diatas dilakukan sebanyak jenis karakter yang ada pada teks *input*, sehingga maksimum perulangan yang dapat dilakukan oleh proses di atas adalah 256 kali.

2. Proses pendekodean. Proses pendekodean dapat dilihat pada algoritma di bawah ini:

```
1   Baca 8 bit pertama dari kode input
2   IndexToSave ← 0
3   While IndexToSave < TotalKarakter {
                    (tag – l + 1)* Total Karakter – 1
4   value ←  ------------------------------------
                          h – l + 1
5   for i = 1 until Total jenis karakter
      ‘mencari interval untuk memperoleh karakter
6     if ( Batas bawah karakter ke-i <= value <  Batas atas
      karakter ke-i ) Then
7       FileSave(IndexToSave) ← karakter ke-i
8       Else
9           i ← i + 1
      Next
                    (h – l + 1) × BatasBawah karakter ke-i
10  l ← l +  ------------------------------------------
                              TotalKarakter
                    (h – l + 1) × BatasAtas karakter ke-i
11  h ← l +  ------------------------------------------ – 1
                              TotalKarakter
12  Do
13        If  h < Nilai_Tengah then
14          l ← l * 2
15          h ← h * 2 + 1
16          t  ← t * 2 + bit selanjutnya dari input
17      Else if l >= Nilai_Tengah then
18          l ← 2 * (l - Half)
19          h ← 2 * (h - Half) + 1
20          t  ← 2 * ( t – Half) + bit selanjutnya dari input
21      Else if l >= Kuartil Pertama dan h
        <=Kuartil_Ketiga Then
22          l ← 2 * (l – Kuartil Pertama)
23          h ← 2 * (h – Kuartil Pertama) + 1
24          t  ← 2 * ( t – Kuartil Pertama) + bit
            selanjutnya dari input
25      Else
26          Exit Do
    Loop
27  IndexToSave ← IndexToSave + 1
    }
```

Baris 4 dilakukan sebanyak total karakter dari teks i*nput.* Proses pencarian interval pada baris 7 akan dilakukan sebanyak total jenis karakter dan maksimum sebanyak 256 kali sampai menemukan interval yang tepat. Untuk penghitungan nilai *l* dan *h* pada baris 10 dan 11 akan dilakukan sebanyak total karakter dari teks *input*. Sedangkan untuk baris 13 sampai 26, akan dilakukan maksimum sebanyak 30 kali. Secara keseluruhan, kompleksitas dari proses pendekodean adalah *O(n)* dengan *n* banyaknya karakter dari teks *input*.

KESIMPULAN DAN SARAN

Kesimpulan

Semakin banyak ragam karakter yang terdapat pada teks *input* maka rasio kompresi yang dihasilkan akan semakin rendah, untuk beberapa kasus *file* kompresi akan memiliki ukuran yang lebih besar daripada *file* sumber. Hasil terbaik diberikan oleh *file input* yang memiliki karakter dengan peluang mendekati satu, semakin besar ukuran *file* tersebut akan semakin tinggi rasio kompresi yang dihasilkan.

Saran

Teknik kompresi pengkodean aritmetika yang diimplementasikan pada program ini adalah pengkodean aritmetika statis. Untuk selanjutnya, dapat dicoba menggunakan pengkodean aritmetika semi-adaptif dan adaptif, kemudian dibandingkan untuk mengetahui performa masing-masing metode tersebut. Penelitiain yang dilakukan terbatas pada *file* teks, untuk pengembangan selanjutnya dapat dicoba untuk diterapkan pada *file* dengan format lain, misalnya gambar.

DAFTAR PUSTAKA

Chrochemore, M & T. Lecroq. 2000. *Text data compression algorithms*.http://citeseer.nj.nec.com/19499.htm. [18 April 2003]

Howard, Paul G. & Vitter, Jeffrey Scott. 1992. *Practical Implementation of Arithmetic Codings.* http://citeseer.nj.nec.com/howard92practical.htm. [18 April 2003]

Nelson, Mark. 1991. *Arithmetic Coding + Statistical Coding = Data Compression*. www.dogma.net/markn/articles/arith/part1.htm. [18 April 2003]

Lelewer, Debra. A & Hirschberg, Daniel. S. 1987. *Data Compression*. http://citeseer.nj.nec.com/lelewer87.htm. [18 April 2003]

Sayood, Khalid. 2000. *Introduction to Data Compression*, *2nd Ed*. Morgan Kaufmann Publisher. San Fransisco, CA

Campos, Arturo San Emeterio. 1999. *Arithmetic Coding*.http://www.arturocampos.com [18 April 2003]