# PERBANDINGAN ALGORITMA HUFFMAN STATIK DENGAN ALGORITMA HUFFMAN ADAPTIF PADA KOMPRESI DATA TEKS


Bib Paruhum Silalahi[1], Julio Adisantoso[2], Danny Dimas Sulistio[3]

[1]Departemen Matematika, FMIPA, Institut Pertanian Bogor
Jl. Raya Pajajaran Bogor, Indonesia

[2]Departemen Ilmu Komputer, FMIPA, Institut Pertanian Bogor
Jl. Raya Pajajaran Bogor, Indonesia

[3]Departemen Ilmu Komputer, FMIPA, Institut Pertanian Bogor
Jl. Raya Pajajaran Bogor, Indonesia

Email: ediary@plasa.com



## ABSTRAK

*Penelitian ini bertujuan untuk mempelajari dan membandingkan unjuk kerja dari algoritma Huffman Statik dan algoritma Huffman Adaptif pada kompresi data. Ruang lingkup penelitian hanya terbatas pada kompresi data teks (*.txt) yang dilakukan pada tiga buah percobaan yaitu percobaan dengan menggunakan file teks yang berasal dari potongan artikel, percobaan dengan menggunakan file teks dengan satu variasi karakter, dan percobaan menggunakan file teks dengan lima dan 256 variasi karakter. Berbagai kriteria yang digunakan dalam perbandingan kedua algoritma diantaranya rasio kompresi, lamanya waktu yang diperlukan untuk mengkompresi file, dan lamanya waktu untuk mendekompresi file menjadi seperti semula.Kompresi menggunakan algoritma Huffman Statik memiliki kompleksitas sebesar O(n lg m), sedangkan algoritma Huffman Adaptif memiliki kompleksitas sebesar O(n.m), dengan nilai n adalah banyaknya karakter dan m adalah besarnya variasi karakter. Dari percobaan potongan artikel menunjukan waktu iterasi yang diperlukan oleh algoritma Huffman Statik untuk melakukan kompresi dan dekompresi adalah cenderung lebih kecil dibandingkan dengan yang dilakukan oleh algoritma Huffman Adaptif. Namun untuk hasil kompresi terlihat unjuk kerja Huffman Adaptif adalah lebih baik dibandingkan Huffman Statik.*



Kata kunci: algoritma, Huffman Statik, Huffman Adaptif, kompleksitas, kompresi data.


## PENDAHULUAN

Pada saat ini kebutuhan akan informasi sangat diperlukan oleh masyarakat umum. Setiap informasi yang beredar dapat direpresentasikan dalam bentuk citra, suara maupun teks secara digital. Dengan semakin banyaknya informasi yang perlu disimpan secara digital, secara automatis akan meningkatkan keperluan untuk menyediakan media penyimpanan data yang lebih besar lagi. Oleh karena itu diperlukan suatu alternatif mekanisme penyimpanan data sehingga dengan media penyimpanan yang ada, dapat menyimpan data sebanyak-banyaknya.

Pemampatan data, atau yang lebih dikenal dengan istilah kompresi data merupakan salah satu metode untuk memperkecil kebutuhan penyimpanan data pada suatu media penyimpanan data. Dengan melakukan metode kompresi dapat memperkecil ukuran data, sehingga kebutuhan akan media penyimpanan data dapat lebih efektif dan ukuran data yang disimpan dapat optimal. Selain berguna dalam penyimpanan data, kompresi data dapat membantu memperkecil ukuran data yang ditransmisikan di dalam suatu media jaringan, seperti internet. Sehingga dengan ukuran data yang lebih kecil, maka waktu yang diperlukan untuk mentransfer data tersebut dapat diperkecil. Namun dibalik kelebihan yang ditawarkan oleh metode ini, harus diperhatikan pula lamanya waktu yang diperlukan untuk membuat data menjadi termampatkan, dan lamanya waktu untuk mengembalikan data tersebut seperti sediakala.

Pada penelitian kali ini akan dipaparkan suatu kajian algoritma Huffman melalui metode statik dan metode adaptif. Algoritma Huffman merupakan salah satu pelopor lahirnya kompresi data, sehingga ukuran data yang perlu disimpan menjadi lebih kecil dibandingkan dengan ukuran data sebenarnya.

Penelitian mengenai kompresi terhadap data teks menggunakan algoritma Huffman telah dilakukan sebelumnya oleh Hutasoit [3] mengenai pengaruh n-gram dalam pembentukan kode Huffman pada *file* teks berbahasa Indonesia dan Layungsari [4] mengenai implementasi kompresi multi tahap menggunakan algoritma Huffman pada *file* teks. Kesimpulan dari penelitian Hutasoit [3] adalah pengaruh n-gram dapat menghasilkan rasio kompresi yang lebih baik yang ditunjukkan dengan rasio kompresi untuk *tree* yang monogram lebih kecil dibandingkan dengan *tree* digram. Dari penelitian Layungsari [4] dapat disimpulkan bahwa hasil kompresi *file* teks menggunakan kompresi multi tahap Huffman menunjukan hasil yang tidak memuaskan. Hal ini dikarenakan *file* hasil kompresi oleh kompresi tahap pertama telah menghasilkan kode prefiks yang optimal.

*Output* dari penelitian ini adalah sebuah sistem yang dapat membandingkan hasil kompresi yang dilakukan oleh metode Huffman Statik dengan hasil kompresi yang dilakukan oleh metode Huffman Adaptif. Selain itu diharapkan dengan pengembangan metode ini dapat dijadikan acuan lebih lanjut dalam penelitian mahasiswa mengenai metode pemampatan *file*.

## TINJAUAN PUSTAKA

### Kompresi Data

Kompresi data dapat dilihat sebagai kumpulan teori informasi dengan tujuan utamanya adalah untuk memperkecil ukuran data yang akan ditransmisikan [5]. Secara sederhana, karakteristik dari kompresi data dapat dianalogikan sebagai sebuah proses untuk mengubah sebuah *string* yang merupakan kumpulan karakter menjadi sebuah *string* yang baru dengan informasi yang sama namun dengan lebar atau ukuran yang lebih kecil. Suatu cara untuk mengembalikan data yang telah terkompresi menjadi seperti sediakala dikenal dengan istilah dekompresi data.

Secara garis besar kompresi data dapat dikelompokkan ke dalam dua metode yaitu metode kompresi *lossy* dan metode kompresi *lossless*. Metode kompresi *lossy* adalah suatu metode kompresi data dengan data yang telah terkompresi apabila dikembalikan ke dalam bentuk semula akan terdapat beberapa informasi yang hilang. Contoh dari metode ini adalah metode yang digunakan pada pemampatan data gambar dan data suara. Sedangkan untuk metode kompresi *lossless* merupakan suatu metode kompresi data dengan data yang telah terkompresi akan dapat dikembalikan seperti semula tanpa ada informasi yang hilang. Implementasi dari metode ini yaitu pada pemampatan data teks atau dokumen ASCII.

Pada metode kompresi *lossless* terdapat dua buah model utama yaitu metode *lossless* dengan menggunakan model statistik dan model kamus. Pada model statistik, kompresi data diawali dengan melakukan perhitungan setiap karakter yang ada di dalam *file*, kemudian dengan statistik karakter yang ada akan dilakukan pengkodean karakter dengan representasi lain. Representasi tersebut apabila dibandingkan dengan karakter asli diharapkan akan lebih kecil ukurannya. Model ini digunakan pada algoritma Huffman dan algoritma Aritmatik.

Sedangkan untuk model kamus, kompresi data diawali dengan melakukan perhitungan setiap *string* yang terdapat di dalam *file*. Kumpulan *string* tersebut akan disusun seperti sebuah indeks dan akan disimbolkan dengan sebuah representasi yang unik. Model ini digunakan pada algoritma Lempel-Ziv dan turunannya (LZW,LZSS, LZRW, dsb).

### Algoritma Huffman

Algoritma Huffman diperkenalkan pertama kali oleh D.A. Huffman pada tahun 1950. Ide dasar dari algoritma ini adalah membuat kode dengan representasi *bit* yang lebih pendek untuk karakter ASCII yang sering muncul di dalam *file* dan membuat kode dengan representasi *bit* yang lebih panjang untuk karakter ASCII yang jarang muncul di dalam *file* [2]. Dalam perkembangannya algoritma Huffman terpecah menjadi dua buah kategori yaitu:

- Algoritma Huffman Statik adalah suatu algoritma yang menggunakan kemungkinan kemunculan dari setiap karakter yang telah ditetapkan pada awal pengkodean dan kemungkinan kemunculan karakter tersebut juga dapat diketahui oleh baik oleh enkoder maupun dekoder.
- Algoritma Huffman Adaptif adalah suatu algoritma dengan kemungkinan kemunculan dari setiap simbol tidak dapat ditentukan dengan pasti selama pengkodean. Hal ini disebabkan oleh perubahan pengkodean secara dinamis berdasarkan frekuensi dari simbol yang telah diolah sebelumnya.

### Algoritma Huffman Statik

Cara kerja dari algoritma Huffman Statik untuk mengkompresi data yaitu dengan cara sebagai berikut:

1. Hitung jumlah karakter yang muncul di dalam *file*, sehingga masing-masing karakter yang ada akan memiliki bobot sesuai dengan banyaknya karakter tersebut di dalam *file*. Kemudian susunlah suatu pohon biner yang dibangun berdasarkan bobot karakter yang ada. Inti dari pembangunan pohon biner adalah menggabungkan dua buah karakter dengan tingkat kemunculan (bobot) yang lebih kecil, kemudian membangkitkan satu buah node *parent* yang memiliki bobot gabungan dari kedua karakter tersebut.
2. Lakukan pengkodean (*encoding*) karakter yang ada di dalam *file* menjadi suatu representasi *bit* sesuai dengan urutan pada pohon biner yang telah dibangun.

Algoritma dekompresi Huffman Statik dapat dijabarkan sebagai berikut:

```
1. Susun pohon biner.
2. X=""
3. Lakukan sampai karakter terakhir
   {selama X bukan leaf_Tree
      {X ← X + bit_selanjutnya}
    Kirim X ke buffer_file_dekompresi
    X=""
    }
4. Simpan   buffer_file_dekompresi   ke
   dalam file tujuan
```

### Algoritma Huffman Adaptif

Algoritma Huffman Adaptif merupakan pengembangan lebih lanjut dari algoritma Huffman. Ide dasar dari algoritma ini adalah meringkas tahapan algoritma Huffman tanpa perlu menghitung jumlah karakter keseluruhan dalam membangun pohon biner. Algoritma ini dikembangkan oleh trio Faller, Gallager, dan Knuth dan kemudian dikembangkan lebih lanjut oleh Vitter, sehingga tercipta dua buah algoritma Huffman Adaptif yaitu algoritma FGK dan algoritma V. Secara umum algoritma Huffman Adaptif adalah sebagai berikut:

1. Prosedur enkoder

```
initialize_model();
do{
        c=getc(input);
        encode(c,output);
        update_model(c);
  }while(c!=eof);
```

2. Prosedur dekoder

```
initialize_model();
while((c=decode(input))!=eof);
  {
        putc(c,output);
        update_model(c);
  }
```

Dapat dilihat dari di atas, pada saat data akan dikompresi, pertama kali akan diinisialisasi sebuah pohon biner dengan sebuah node yang dikenal dengan *Not-Yet-Transmitted* (NYT) atau kode *escape*. Kemudian akan dilakukan pembacaan karakter satu persatu dari awal *file* sampai akhir. Selama pembacaan karakter, akan dikirim kode NYT setiap terdapat node (karakter) baru yang belum ada di dalam pohon. Hal ini akan memudahkan dekoder untuk membedakan antara sebuah kode dengan sebuah karakter baru. Setelah kode tersebut dikirimkan bersama kode ASCII karakter baru, maka diperlukan penambahan sebuah node baru untuk karakter baru, dan sebuah kode NYT dari kode NYT yang lama. Dan terakhir akan dilakukan peng-*update*-an pohon.

Sedangkan untuk proses dekompresi pada prosedur dekoder, tidak jauh berbeda dengan proses kompresi data. Hal yang berbeda hanya pada saat membaca data dari *file* tidak dilakukan perkarakter, namun dibaca per-*bit*, hal ini untuk mengidentifikasi keberadaan karakter yang diinisialisasi pada pohon, apakah sudah ada atau belum. Jika belum ada akan dilakukan penambahan node baru, namun apabila sudah ada akan dilakukan perubahan bobot karakter. Setelah itu diakhiri dengan peng-*update*-an pohon.

Salah satu kelebihan dari algoritma kompresi Huffman Adaptif ini adalah kemampuannya untuk melakukan kompresi *file* tanpa perlu mengetahui jumlah frekuensi dari masing-masing karakter sehingga tidak menyimpan *tree* pada *file* hasil kompresi.

## METODOLOGI

### Pengumpulan Bahan

Data yang digunakan adalah data teks dengan ekstensi *.txt dengan berbagai rentang ukuran *file*. Berbagai rentang ukuran *file* yang digunakan yaitu < 20.000 *byte*, antara 20.000 sampai 40.000 *byte* dan *file* dengan ukuran > 40.000 *byte*. *File* teks tersebut berisi potongan tajuk rencana harian sebuah media cetak *online* (Media Indonesia) yang beredar antara bulan Juni sampai November 2003. Selain itu digunakan sebuah mekanisme untuk membangkitkan karakter dengan rentang bervariasi dari satu buah variasi karakter.

### Struktur Percobaan

Dalam penelitian ini dilakukan beberapa pengujian yang antara lain:

1. Melihat pola rasio pemampatan dan lamanya kompresi dan dekompresi algoritma dengan menggunakan *file* yang berasal dari potongan artikel.
2. Melakukan simulasi dengan menggunakan *file* teks yang hanya memiliki satu buah variasi karakter kemudian dilihat pola rasio pemampatan dan lamanya eksekusi dalam rentang 1.000 *byte*, 2.000 *byte*, 3.000 *byte* sampai 100.000 *byte*.
3. Mengamati rasio dan lamanya eksekusi algoritma pada *file* yang memiliki lima buah variasi karakter yang berbeda, dan 256 variasi karakter yang berbeda.
4. Penarikan kesimpulan menggunakan uji Analisis Ragam (ANOVA).

### Analisis

Percobaan yang telah dirancang selanjutnya diuji coba dan dilakukan analisis terhadap kinerja algoritma dengan berbagai kriteria berikut:

1. Kebutuhan tempat penyimpanan

   Seluruh *file* yang diuji perlu dicatat ukuran *file* semula dan ukuran *file* setelah dikompresi. Hal ini diperlukan untuk mendapatkan rasio pemampatan yang dilakukan oleh algoritma tertentu. Perhitungan rasio pemampatan dapat didefinisikan sebagai berikut:

$$\text{Rasio Pemampatan} = \frac{\text{loss}}{\text{ukuran file asli}} \qquad (1)$$

   dengan:

   *loss* =ukuran *file* awal–ukuran *file* kompresi
   =ukuran *file* yang hilang akibat termampatkan.

2. Pengamatan *running time* program

   Kompleksitas dari suatu algoritma dapat diestimasi dengan melakukan perhitungan *running time*. Dengan melihat *flowchart* masing-masing algoritma, sehingga dapat dihitung kompleksitas dari masing-masing algoritma tersebut.

### Perancangan Sistem

- Sistem dapat melakukan kompresi dan dekompresi *file* pada kedua algoritma.
- Dapat menampilkan statistik hasil kompresi dan dekompresi dalam bentuk data dan grafik.

**Desain Sistem**

Sebelum mengimplementasikan rancangan sistem, perlu ditentukan dahulu alur sistem sehingga dapat menunjang kinerja sistem yaitu:

1. Desain masukan

   Pada masing-masing rentang *file* akan berisi 10 buah *file* teks sumber (artikel) yang akan digunakan pada saat pengujian.
2. Desain keluaran

   Untuk membedakan antara *file* teks dengan *file* hasil kompresi dan dekompresi dari masing-masing algoritma, maka diberikan ekstensi *file* yang berbeda untuk setiap operasi yaitu:
   - Algoritma Huffman Statik

     *File* hasil kompresi menggunakan ekstensi *.huf, sedangkan *file* hasil dekompresi menggunakan ekstensi *.new.
   - Algoritma Huffman Adaptif

     *File* hasil kompresi menggunakan ekstensi *.ada, sedangkan *file* hasil dekompresi menggunakan ekstensi *.now.
3. Desain Proses

   *File* artikel yang telah dikelompokkan ke dalam beberapa rentang akan dipisahkan ke dalam beberapa direktori (misal 1,2, dan 3). Setiap *file* tersebut akan diujikan sebanyak 10 kali ulangan. Pada saat pengujian dan simulasi, hasil dari kompresi dan dekompresi akan dimasukkan ke dalam sebuah *file* teks (*.txt) secara automatis yang kemudian akan diolah dan digabungkan ke dalam *file* Excell (*.xls) secara manual. Kemudian diuji dengan ANOVA menggunakan program SAS versi 8.

## HASIL DAN PEMBAHASAN

Untuk mempemudah pengaksesan data, rentang *file* yang telah didefinisikan sebelumnya kemudian direpresentasikan sebagai tiga buah sub direktori (1, 2, dan 3) dengan masing-masing sub direktori tersebut memiliki 10 buah *file* teks yang diberi nama 1.txt, 2.txt, 3.txt sampai 10.txt.

Untuk mendapatkan hasil kompresi dilakukan dengan cara memilih menu simulasi kompresi dan untuk mendapatkan hasil dekompresi dilakukan dengan memilih menu simulasi dekompresi. Kedua menu ini dipecah berdasarkan masing-masing algoritma.

Setelah simulasi kompresi dan dekompresi dijalankan, pada setiap direktori yang memiliki sub direktori akan dihasilkan *file* teks yang berisi rangkuman rasio pemampatan, waktu eksekusi dan identitas *file* yang diuji. Dengan bantuan aplikasi Ms Excell penulis mengolah data pada *file* teks tersebut, kemudian mencari rata-rata dari masing-masing kriteria uji.

**Representasi Struktur Data**

1. Pada *file* kompresi Huffman Statik

   *File* kompresi (dengan ekstensi *.huf) diawali dengan *header file* Huffman dengan panjang 4 *byte* yang menunjukan *file* tesebut dikompresi atau tidak. Kemudian disambung 1 *byte* sebagai kode CRC sederhana. Lalu disambung dengan 4 *byte* untuk menyimpan informasi ukuran *file* sumber, 2 *byte* untuk menyimpan banyaknya variasi karakter, yang kemudian disambung dengan pasangan-pasangan 2 *byte* yang berisi karakter dan panjang representasi karakter tersebut pada pohon Huffman. Dan pada akhirnya akan dituliskan representasi pohon biner dan hasil *encoding file* sumber menggunakan pohon biner.
2. Pada *file* kompresi Huffman Adaptif

   *File* kompresi (dengan ekstensi *.ada) diawali dengan kode *escape*, disambung representasi kode *escape* pada pohon biner. Setiap kali ada karakter baru, akan dikirimkan terlebih dahulu representasi kode *escape* yang disambung dengan kode ASCII yang bersangkutan. Sedangkan untuk karakter yang telah didefinisikan, maka akan disimpan representasi karakter yang bersangkutan.
3. Pada *file* statistik simulasi artikel

   Diawali dengan judul untuk kolom-kolom yang ada seperti rasio pemampatan, lama eksekusi, alamat *file* sumber, ukuran *file* sumber, alamat *file* tujuan, ukuran *file* tujuan dan pada baris baru akan disambung dengan data *file* yang berada pada sub direktori yang sedang diuji.

**Kompleksitas Algoritma Huffman Statik**

Secara garis besar alur algoritma Huffman Statik pada saat mengkompresi *file* adalah:

```
1. Hitung banyak karakter yang ada
   pada file.
2. Lakukan sorting jumlah_Karakter
   yang ada.
3. Bentuk pohon Huffman.
4. Ubah seluruh karakter yang ada di
   dalam file sesuai dengan
   representasi bit pada pohon.
```

Kompleksitas pada iterasi 1 prosedur kompresi Huffman di atas adalah O(n) dengan n adalah besarnya ukuran dari *file* yang merepresentasikan banyaknya karakter yang ada dalam *file* tersebut. Sedangkan pada iterasi 2 memiliki kompleksitas sebesar O(n lg n) dengan n adalah banyaknya karakter yang muncul. Misal terdapat sebuah *string* aabccca maka n untuk iterasi 2 adalah tiga (karakter yang muncul a, b, dan c).

Pada iterasi 3, pembentukan pohon Huffman yaitu dengan cara mengambil 2 buah node dari array yang sudah terurut pada iterasi 2 dengan bobot kemunculan terkecil, kemudian dibuat sebuah node dengan bobot gabungan dari kedua node tersebut. Lalu keluarkan kedua node dari *array*, dan masukkan node baru ke dalam *array*. Prosedur ini terus berulang sampai tersisa hanya 1 buah node pada array yang memiliki bobot yang merepresentasikan jumlah seluruh karakter yang ada di dalam *file*. Oleh karena pembentukan pohon ini menggunakan kaidah algoritma Heap [1], sehingga kompleksitas dari iterasi 3 ini adalah O(n lg n). Kode semu untuk prosedur pembentukan pohon Huffman adalah sebagai berikut:

```
Untuk semua node dalam Array
Cari 2 buah node dengan bobot terkecil
dari Array;
  {
    node1=lowest(Array);
    node2=2nd_lowest(Array);
  }
Buat node baru dengan bobot gabungan,
dan node baru tersebut menjadi parent
dari kedua node terkecil
 {
     NewNode.weight=node1.weight+
                    node2.weight;
     NewNode.LeftChild =node1;
     NewNode.RightChild =node2;
     node1.Parent=NewNode;
     node2.Parent=NewNode;
 }
```

Pada iterasi 4, akan dilakukan proses transformasi seluruh karakter yang ada di dalam *file*. Sebelum mengubah karakter, terlebih dahulu didefinisikan nilai biner dari cabang pohon Huffman yang telah terbentuk, seperti berikut:

```
Create_Bit_Sequence(From_Top            to
Every_Leaf)
    {
  Untuk Node yang berada di sebelah
  kiri diberi nilai 0 dan untuk Node
  yang berada disebelah kanan diberi
  nilai 1. Pemberian nilai ini
  dilakukan mulai dari cabang paling
  atas sampai pada seluruh ujung daun.
 }
```

Setelah itu barulah dilakukan proses *encoding* seluruh karakter *file* sumber, seperti berikut:

```
  Lakukan untuk seluruh byte yang ada
  di dalam file
    { Baca 1 byte, representasikan
    dalam bentuk karakter ASCII. Baca
    representasi karakter dalam pohon
    Huffman.       Masukan setiap bit
    representasi satu persatu kedalam
    buffer array yang berukuran 8.
    Jika buffer array terisi penuh,
    bentuk sebuah karakter yang sesuai
    nilai array, lalu set array
    menjadi kosong. }
   Untuk byte terakhir, jika array
   buffer masih ada nilai, bentuk
   sebuah karakter yang sesuai nilai
   array, lalu set array menjadi
   kosong.
```

Kompleksitas algoritma untuk prosedur di atas adalah O(n) dengan n merupakan besarnya variasi karakter, karena pada iterasi ini dilakukan kunjungan dari node yang paling atas sampai seluruh node dilewati, sedangkan pada Gambar 8 memiliki kompleksitas O(n lg m) dengan n adalah besar ukuran *file* dan m adalah besar variasi karakter. Dari uraian di atas dapat disimpulkan bahwa untuk kompresi menggunakan algoritma Huffman Statik memiliki kompleksitas sebesar O(n lg m), dengan nilai n adalah banyaknya karakter dan m adalah besarnya variasi karakter.

**Kompleksitas Algoritma Huffman Adaptif**

Algoritma Huffman Adaptif secara umum dapat dituliskan sebagai berikut:

```
BEGIN
 1.Inisialisasi tree
 'untuk seluruh karakter yang ada di
  dalam file
     {
     2.Baca karakter satu-satu
     3.Ambil representasi karakter
       dari pohon huffman
     4.Jika representasi tersedia,
       kirim bit ke stream
     5.Jika belum tersedia, tambahkan
       karakter yang bersangkutan ke
       stream
     6.Lakukan update bobot karakter
       yang terkirim pada pohon
       Huffman
     }
 7.Kirim bit ke stream untuk escape
   code
 8.Jika stream masih memiliki nilai,
   tambahkan digit 0 sebanyak sisa
   ruang yang masih kosong
 9.Bentuk karakter dari stream
   terakhir
END
```

Pada iterasi 1, dilakukan inisialisasi *tree* dengan masing-masing node belum memiliki bobot dan belum menunjuk ke mana-mana. Kompleksitasi iterasi ini adalah O(n) dengan n sebesar 512. Hal ini didasari bahwa maksimum banyaknya node pada *tree* adalah sebesar 2 x (256) - 1 yaitu sebanyak 511 node. Adapun prosedur inisialisasi *tree* sebagai berikut:

```
BEGIN
    'set node tidak memiliki hubungan
    satu dengan yang lain dan belum
    memiliki bobot
    'set posisi node belum ada di
    dalam tree
    'set node yang terkirim = 0
    'Update_Tree(EscapeCode)
END
```

Setelah sebuah karakter dibaca oleh mesin enkoder, akan dilakukan pengecekan tentang sudah ada atau belum karakter tersebut di dalam *tree*. Jika representasi karakter sudah ada, maka representasi *bit* karakter tersebut akan dimasukan ke dalam buffer. Namun jika belum, maka nilai representasi *bit* dari node NYT yang akan disimpan. Prosedur untuk penyimpanan *bit* adalah:

```
BEGIN
    'input representasi karakter
    kedalam buffer dalam satuan bit
    'jika buffer penuh,masukkan nilai
    buffer kedalam senarai data, lalu
    set buffer=0
END
```

Kompleksitas untuk prosedur di atas adalah sebesar O(lg n), dengan n adalah besar variasi karakter yang sudah diolah oleh mesin enkoder. Setelah penyimpanan nilai *bit* karakter, akan dilakukan pengubahan pohon Huffman, dengan cara menambahkan bobot karakter yang dibaca, yang disambung dengan penambahan nilai bobot *parent* sampai node yang paling atas, seperti berikut:

```
BEGIN
    'baca posisi karakter di dalam
    tree
    'jika karakter belum ada, maka
    tambahkan node baru kedalam tree,
    lalu set:
        'bobot=1,      parentnode=NYT,
        lalu set NYT berada pada
        RightNode NYT lama
    'jika karakter sudah ada, maka
    tambahkan bobot karakter pada
    tree
    'lakukan pertukaran node jika
    diperlukan mulai dari posisi
    karakter sampai ke root
END
```

Kompleksitas untuk proses *update tree* di atas adalah sebesar O(n) dengan n adalah besar variasi karakter yang terbaca. Dari uraian di atas dapat disimpulkan bahwa untuk kompresi menggunakan algoritma Huffman Adaptif memiliki kompleksitas sebesar O(nm), dengan nilai n adalah banyaknya karakter dan m adalah besarnya variasi karakter.

**Percobaan Artikel Editorial**

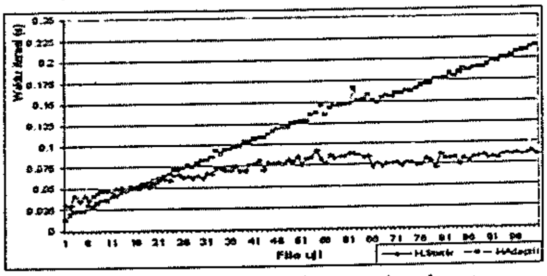
Tujuan dari percobaan artikel ini adalah untuk mendapatkan rasio dan lamanya waktu eksekusi dari kedua algoritma pada tiga buah rentang *file* potongan artikel yaitu percobaan pada rentang *file* < 20.000 *byte*, rentang *file* antara 20.000 sampai dengan 40.000 *byte*, dan percobaan pada rentang *file* > 40.000 *byte*. Statistik rata-rata lamanya iterasi kompresi, rasio kompresi, dan lamanya iterasi dekompresi yang dilakukan oleh masing-masing algoritma dapat dirangkum pada Tabel 1. Dapat dilihat bahwa waktu iterasi untuk kompresi dan dekompresi Huffman Statik lebih cepat dibandingkan waktu iterasi kompresi dan dekompresi Huffman Adaptif. Namun rasio pemampatan Huffman Adaptif lebih besar dibandingkan rasio pemampatan Huffman Statik. Hal ini menjadikan *file* kompresi yang dihasilkan oleh Huffman Adaptif akan lebih kecil ukurannya dibandingkan dengan *file* kompresi yang dihasilkan oleh Huffman Statik. Selain itu dapat terlihat bahwa semakin besar ukuran *file* yang akan dikompresi berimplikasi pada lama iterasi yang semakin lama.

**Tabel 1.** Rata-rata lama dan rasio kompresi *file* artikel

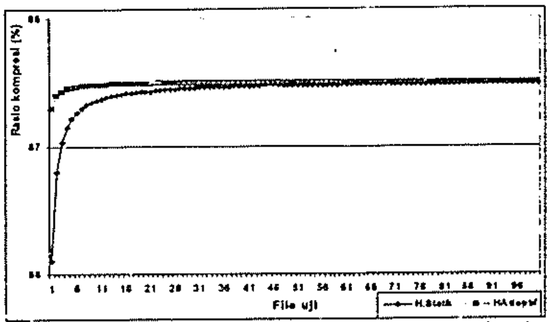
| | Ukuran *file* | Kompresi | | Dekompresi | |
|---|---|---|---|---|---|
| | | Lama(s) | Rasio(%) | Lama(s) | Rasio(%) |
| Huffman Statik | < 20 KB | 0,0564441 | 42,6445 | 0,03934281 | -74,363 |
| | 20 s/d 40 KB | 0,0656619 | 43,1081 | 0,04878656 | -75,777 |
| | > 40 KB | 0,0763654 | 43,2218 | 0,05559875 | -76,13 |
| Huffman Adaptif | < 20 KB | 0,1235163 | 43,3365 | 0,11193219 | -76,493 |
| | 20 s/d 40 KB | 0,2234394 | 43,4659 | 0,18995875 | -76,889 |
| | > 40 KB | 0,3226191 | 43,4665 | 0,26700437 | -76,893 |

**Percobaan Satu Variasi Karakter**

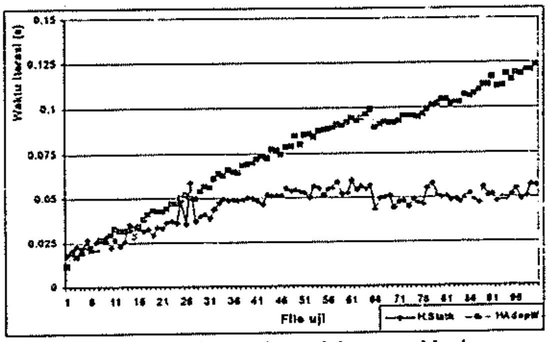
Tujuan dari percobaan ini untuk mendapatkan rasio dan lamanya waktu eksekusi dari kedua algoritma untuk *file* yang memiliki karakter dengan peluang kemunculan adalah satu Hasil iterasi kedua algoritma pada saat ditampilkan akan dibedakan dengan warna, pada hasil iterasi algoritma Huffman Statik ditunjukkan dengan warna hitam dan untuk hasil algoritma Huffman Adaptif ditunjukkan dengan warna abu tua. Dapat dilihat dari Gambar 1 dan 3 bahwa lama iterasi untuk kompresi dan dekompresi Huffman Statik untuk satu variasi karakter cenderung lebih cepat dibandingkan dengan Huffman Adaptif, walaupun pada rentang *file* 1.000 *byte* sampai 20.000 *byte* masih terjadi fluktuasi disebabkan faktor ukuran *file* yang terlalu kecil dan iterasi yang cukup cepat. Pada Gambar 2 dapat terlihat bahwa rasio kompresi Huffman Adaptif lebih baik dibandingkan rasio kompresi Huffman Statik, dengan kecenderungan semakin stabil seiring besarnya ukuran *file*.



**Gambar 1.** Perbandingan lama kompresi pada satu karakter.




Gambar 2. Perbandingan rasio kompresi kedua algoritma pada satu karakter.




Gambar 3. Perbandingan lama dekompresi kedua algoritma pada satu karakter.

## Percobaan Lima dan 256 Variasi Karakter

Tujuan dari percobaan ini adalah mengamati rasio dan lamanya eksekusi algoritma pada *file* dengan lima buah variasi karakter yang berbeda, dan 256 buah variasi karakter yang berbeda. Pada kompresi *file* yang berisi lima karakter terjadi suatu anomali dapat dilihat pada Tabel 2, menunjukan hasil kompresi yang seharusnya mengecil menjadi membesar. Pada algoritma Huffman Statik disebabkan oleh penambahan karakter diawal *file*, berupa *header* terkompresi tidaknya *file* (4 *byte*), 1 *byte* kode CRC, 4 *byte* untuk menyimpan ukuran *file* sumber, dan 2 *byte* untuk menyimpan banyaknya karakter. Ditambah kebutuhan untuk menyimpan *tree* berupa 2 *byte* untuk masing-masing karakter, disambung representasi *tree* dan representasi hasil kompresi. Sedangkan pada algoritma Huffman Adaptif mengalami pembesaran ukuran *file* karena selain menyimpan setiap variasi karakter yang muncul, akan disimpan pula hasil pengkodean berdasarkan *tree* . Karena *tree* yang dibentuk pada Huffman Adaptif tidak perlu disimpan, menjadikan rasio kompresi Huffman Adaptif lebih baik dibandingkan rasio kompresi Huffman Statik.

**Tabel 2.** Rata-rata lama dan rasio kompresi *file* berisi lima buah karakter (abcde = 5 *byte*)

| | | |
|---|---|---|
| Huffman Statik | Lama iterasi (detik) | 0,01328 |
| | Rasio pampat (%) | -400 |
| | Ukuran hasil (*byte*) | 25 |
| Huffman Adaptif | Lama iterasi (detik) | 0,008969 |
| | Rasio pampat (%) | -40 |
| | Ukuran hasil (*byte*) | 7 |

Pada percobaan dengan *file* berisi 256 karakter terjadi pula anomali seperti halnya yang terjadi pada percobaan 5 karakter. Seperti yang dapat dilihat pada Tabel 3, rasio kompresi kedua algoritma menunjukan nilai negatif yang berarti hasil kompresi mengalami pembengkakan ukuran. Pada algoritma Huffman Statik lebih disebabkan karena besarnya tempat yang diperlukan untuk menyimpan statistik karakter.

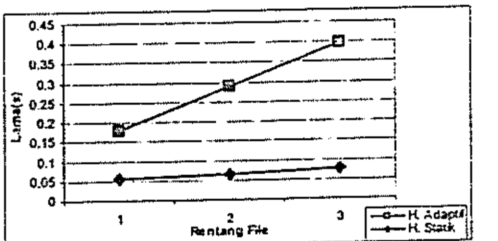
Karena pada percobaan ini digunakan 256 karakter yang berbeda, maka secara automatis akan diperlukan minimal 512 *byte* untuk menyimpan informasi karakter. Hal ini belum termasuk penyimpanan *header*, dan penyimpanan hasil pengkodean *file* sumber. Sedangkan pada algoritma Huffman Statik selain menyimpan 256 *byte* yang merepresentasikan karakter yang berbeda, perlu juga disimpan hasil pengkodean *file* sumber.

**Tabel 3.** Rata-rata lama dan rasio kompresi *file* berisi 256 buah karakter berbeda

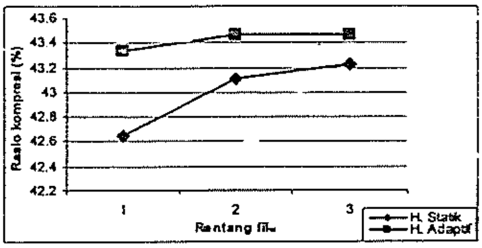
| | | |
|---|---|---|
| Huffman Statik | Lama iterasi (detik) | 0,025 |
| | Rasio pampat (%) | -304,297 |
| | Ukuran hasil (*byte*) | 1035 |
| Huffman Adaptif | Lama iterasi (detik) | 0,011344 |
| | Rasio pampat (%) | -100,781 |
| | Ukuran hasil (*byte*) | 514 |

## Pengujian Menggunakan ANOVA

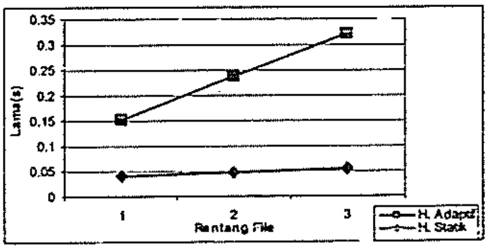
Tujuan dari pengujian ini adalah untuk membantu dalam penarikan kesimpulan akhir dari data-data hasil percobaan dengan menggunakan pendekatan ragam. Kriteria yang diuji diantaranya, lama iterasi kompresi, rasio kompresi, dan lama iterasi dekompresi kedua algoritma. Adapun percobaan yang akan diujikan dengan Anova ini hanyalah percobaan dengan menggunakan potongan artikel. Hal ini dikarenakan representasi karakter yang ada di dalam *file* artikel tersebut biasa digunakan sehari-hari. Dari data pada Tabel 1, dapat ditarik suatu hubungan antara rentang *file* dengan algoritma pada lamanya iterasi kompresi, rasio kompresi dan lamanya iterasi dekompresi dengan menggunakan analisis secara deskriptif seperti pada Gambar 4 sampai Gambar 6.




Gambar 4. Hubungan rentang *file* artikel dengan lamanya iterasi kompresi.




Gambar 5. Hubungan rentang *file* artikel dengan rasio kompresi.




Gambar 6. Hubungan rentang *file* artikel dengan lama iterasi dekompresi.

Dari Gambar 5 dan 6 dapat disimpulkan bahwa lamanya iterasi yang dilakukan oleh algoritma Huffman Adaptif sangat dipengaruhi oleh besar ukuran *file* (rentang *file*). Hal ini berbeda sekali jika dibandingkan dengan algoritma Huffman Statik yang relatif stabil. Dari ketiga gambar di atas dapat ditarik kesimpulan sebagai berikut:

- Iterasi kompresi Huffman Statik lebih cepat 2,19 kali pada rentang 1, lebih cepat 3,4 kali pada rentang 2, dan lebih cepat 4,22 kali pada rentang 3 jika dibandingkan dengan iterasi kompresi Huffman Adaptif.
- Rasio kompresi Huffman Adaptif lebih tinggi 0,69% pada rentang 1, lebih tinggi 0,36% pada rentang 2, dan lebih tinggi 0,24% pada rentang 3 jika dibandingkan dengan rasio kompresi Huffman Statik.
- Iterasi dekompresi Huffman Statik lebih cepat 2,84 kali pada rentang 1, lebih cepat 3,89 kali pada rentang 2, dan lebih cepat 4,8 kali pada rentang 3 jika dibandingkan dengan iterasi kompresi Huffman Adaptif.

Dari perhitungan ANOVA untuk lama iterasi kompresi, rasio kompresi dan lama iterasi dekompresi pada tiga buah rentang *file* artikel didapatkan nilai P yang berada dibawah taraf nyata ($\sigma <= 0,05$ pada selang kepercayaan 95%), sehingga dapat disimpulkan bahwa lamanya waktu proses dan besarnya rasio kompresi dipengaruhi oleh rentang *file* yang ada dan algoritma kompresi yang digunakan. Dari tabel ANOVA tersebut pula dapat disimpulkan bahwa perbedaan waktu dan rasio kompresi pada kedua algoritma berbeda nyata (P-value <= 0,05).

## KESIMPULAN

- Semakin variatif karakter yang muncul, akan memperkecil rasio kompresi yang dihasilkan, baik oleh algoritma Huffman Statik, maupun pada algoritma Huffman Adaptif.
- Rasio kompresi terbaik terjadi pada *file* yang memiliki karakter dengan peluang kemunculan mendekati nilai satu (bobot karakter hampir sebesar ukuran *file*). Rasio terbaik yang didapat selama penelitian yaitu sebesar 87,489%.
- Rasio kompresi terburuk terjadi pada *file* yang memiliki variasi karakter besar dan pada *file* yang berukuran kecil. Hal ini ditandai dengan terjadinya pembesaran ukuran *file* hasil dibandingkan ukuran *file* awal.
- Dengan menggunakan metode kompresi apapun memiliki *trade off*, pengguna akan diberi pilihan hasil maksimum dengan iterasi yang lama atau iterasi yang cepat namun dengan hasil yang biasa.
- Hasil percobaan yang terdapat pada pembahasan menunjukan bahwa waktu iterasi yang diperlukan oleh algoritma Huffman Statik untuk melakukan kompresi dan dekompresi adalah cenderung lebih kecil dibandingkan dengan yang dilakukan oleh algoritma Huffman Adaptif. Namun untuk hasil kompresi terlihat unjuk kerja Huffman Adaptif adalah lebih baik dibandingkan Huffman Statik.

## SARAN

Pada saat mengimplementasikan algoritma Huffman Adaptif, penulis menggunakan prosedur yang menjadikan kompleksitas algoritma Huffman Adaptif menjadi O(n.m). Seharusnya kompleksitas Huffman Adaptif, hampir sama dengan algoritma Huffman Statis yaitu O(n lg m). Pengembangan selanjutnya untuk perbandingan kompresi Huffman Statik dan Huffman Adaptif ini dapat diterapkan pada mekanisme pengiriman paket data terkompresi pada media jaringan.

## DAFTAR PUSTAKA

[1] Cormen, Thomas H., C. E. Leiserson., & R. L. Rivest. 1990. *Introduction to Algorithms*. Mc Graw Hill Company.

[2] Huffman, D. A. 1952. *A Method for the Construction of Minimum-Redundancy Codes**. Proc. IRE, vol. 40, pp. 1098–1101, Sept. 1952.

[3] Hutasoit, Yenny E. 2001. *Huffman Coding Untuk Kompresi Data Teks Berbahasa Indonesia*. Skripsi. Jurusan Ilmu Komputer Fakultas MIPA IPB. Bogor, Indonesia.

[4] Layungsari. 2003. *Penyempurnaan dan Implementasi Software Kompresi Multi Tahap Menggunakan Huffman Coding*. Skripsi. Jurusan Ilmu Komputer Fakultas MIPA IPB. Bogor, Indonesia.

[5] Lelewer, D. A. & Hirschberg, D. S. 1987. *Data Compression*. ACM Computing Surveys 19(1987) 261-296.

[6] Moore, David S. 1994. *The Basic Practice of Statistics*. ISBN 0-7167-2628-9. W.H. Freeman and Company. New York.