# Analisis Perbandingan Kinerja Kombinasi Algoritma *BWT-RLE-MTF-Huffman* Dan *BWT-MTF-RLE-Huffman* Pada Kompresi *File*


Andreas Dony Marhendra S.
Jurusan Informatika
Universitas Sebelas Maret
Jl. Ir. Sutami 36A Surakarta
andomaze@gmail.com

Esti Suryani
Jurusan Informatika
Universitas Sebelas Maret
Jl. Ir. Sutami 36A Surakarta
suryapalapa@yahoo.com

Abdul Aziz
Jurusan Informatika
Universitas Sebelas Maret
Jl. Ir. Sutami 36A Surakarta
Abdul_7773@yahoo.com



## ABSTRACT

*Kompresi data seringkali digunakan untuk proses transmisi data (data transmission) dan penyimpanan data (storage). Keuntungan data yang terkompresi antara lain dapat mengurangi bottleneck pada transmisi data penyimpanan data lebih hemat ruang, mempersulit pembacaan data oleh pihak yang tidak berkepentingan, dan memudahkan distribusi data. Berdasarkan hal tersebut, paper ini menerapkan algoritma kompresi data dengan pengujian pada beberapa file (.txt, .rtf, .doc, .exe, .dll, .tif, dan .bmp). Algoritma yang digunakan merupakan kombinasi beberapa algoritma lossless compression, yaitu Burrows-Wheeler Transform (BWT), Run-Length Encoding (RLE), Move-To-Front (MTF) serta Huffman Coding. Pengujian dilakukan untuk mengetahui besar rasio kompresi, waktu kompresi dan waktu dekompresi. Serta dilakukan perbandingan kinerja pada 2 jenis kombinasi algoritma. Penelitian menghasilkan suatu hasil perbandingan kinerja kompresi file menggunakan kombinasi BMRH (BWT – MTF – RLE – Huffman) dan BRMH (BWT – RLE – MTF – Huffman). Hasil pengujian didapat rata-rata total rasio kompresi, waktu kompresi, dan waktu dekompresi untuk BMRH berturut-turut 70.44 %,* $9.6 \times 10^{-6}$ *second/byte, dan* $22.4 \times 10^{-6}$*, sedangkan BRMH berturut-turut 66.57 %,* $9.2 \times 10^{-6}$ *second/byte, dan* $15.8 \times 10^{-6}$ *second/byte.*



## Keywords

*Burrows-Wheeler*, Kompresi Data, *Huffman*, *Lossless*, *Move-To-Front*, *Run-Length Encoding*.


## 1. PENDAHULUAN

Penggunaan data digital yang semakin merambah di berbagai aspek sering menimbulkan masalah dalam hal ruang penyimpanan dan transmisi data. Oleh karena itu, pengembangan teknik kompresi perlu dilakukan. Pengembangan algoritma kompresi seringkali didasari algoritma yang sudah ada atau berusaha mengkombinasikan dua atau lebih algoritma. Beberapa algoritma dasar yang sering dikembangkan diantaranya adalah *Run-Length Encoding* (RLE) dan *Huffman*.

*Huffman Coding* merupakan algoritma kompresi *lossless* entropi yang menggunakan metode spesifik dengan memanfaatkan frekuensi kemunculan simbol, yang kemudian disajikan dalam suatu *prefix-tree* serta dikodekan dalam kode biner [1]. *Huffman Coding* ini dapat digunakan dalam lingkup yang luas karena kesederhanaan, kecepatan serta fleksibel pada media kompresi apapun juga biasa digunakan sebagai algoritma "*back-end*" pada beberapa metode kompresi yang lainnya [2]. Lain pula dengan *Run-Length Encoding* (RLE), RLE merupakan algoritma kompresi data yang paling sederhana yang memanfaatkan deretan simbol berulang yang panjangnya dikodekan dalam *integer*. Masalah yang muncul dalam RLE terletak pada *worst case* yang dapat menghasilkan *output data* yang lebih besar dari *input data*, untuk meminimalkan masalah tersebut maka dapat dikombinasikan dengan algoritma seperti *Huffman Coding* [3].

Penelitian tentang kompresi data saat ini sering mengkombinasikan algoritma satu dengan yang lainnya untuk mendapatkan performansi kompresi yang lebih tinggi. Michael Dipperstein menyatakan bahwa *Burrows-Wheeler* (BWT) sebagai pendukung dalam kompresi karena BWT mengkonversi data menjadi suatu *format* yang umumnya lebih kompresibel dengan metode algoritma RLE [4]. Dipperstein juga menyatakan algoritma *Move-To-Front* (MTF) dapat meningkatkan kompresi dari metode *statistical encoder* / *entropy* seperti metode dasar *Huffman* atau *Arithmetic* Coding [5]. Bahkan Burrows & Wheeler dalam penelitiannya mencoba melakukan penambahan algoritma MTF setelah algoritma BWT format data yang dihasilkan menjadi lebih kompresibel oleh metode RLE maupun *statistical encoder* (seperti *Huffman*, *Arithmetic*, dan *Shannon-Fano*) bahkan dengan ordo "0" sekalipun [6].

Berdasarkan pemaparan masalah dan pertimbangan dari keunggulan metode-metode tersebut maka pada kesempatan ini penulis mencoba melakukan penelitian dengan mempertimbangkan pernyataan sebelumnya dari Michael Dipperstein [5] dan penelitian dari Telaumbanua [7] mengenai keunggulan kombinasi BWT – RLE dan MTF – *Huffman* kemudian menggabungkan kedua kombinasi tersebut menjadi satu kesatuan yaitu, BRMH (BWT – RLE – MTF – *Huffman*). Penulis juga mengangkat kombinasi kedua dengan mempertimbangkan pernyataan Burrows & Wheeler [6] mengenai keunggulan menggabungkan BWT dan MTF kemudian digabungkan dengan algoritma RLE serta memanfaatkan *Huffman coding* sebagai "*back-end*" dalam kombinasi seperti yang diungkapkan Sharma [2] dan Shanmugasundaram & Lourdusamy [3]. Oleh karena itu penulis mengangkat kombinasi BMRH (BWT – MTF – RLE – *Huffman*) sebagai kombinasi kedua dalam penelitian ini. Kedua kombinasi algoritma tersebut nantinya akan dilakukan pengujian kinerja berdasarkan rasio kompresi, waktu kompresi dan dekompresi untuk mengetahui tingkat efektivitas dalam kompresi kemudian dilakukan analisis perbandingan satu sama lain.

## 2. DASAR TEORI

### 2.1 Kompresi Data

Kompresi data merupakan cabang ilmu komputer yang bersumber dari teori informasi. Teori informasi mengfokuskan pada berbagai metode tentang informasi

termasuk penyimpanan dan pemrosesan pesan. Teori informasi mempelajari pula tentang *redundancy* (informasi tak berguna) pada pesan. Semakin banyak *redundancy* semakin besar pula ukuran pesan, upaya mengurangi *redundancy* inilah yang akhirnya melahirkan subyek ilmu tentang kompresi data [7]. Teknik kompresi data dapat dibagi menjadi dua kategori besar, yaitu :

1. Kompresi *Lossless*

Kompresi *lossless* adalah kompresi yang menjaga keakuratan data dimana perubahan atau hilangnya informasi bahkan beberapa bit saja pada data selama proses kompresi tidak bisa ditoleransi. Sehingga teknik kompresi ini bersifat *reversible* yaitu hasil kompresi bisa dikembalikan ke bentuk semula. Teknik ini lebih cocok diaplikasikan pada file database, teks, gambar medis, atau photo hasil satelit dimana hilangnya beberapa *detail* pada data dapat berakibat fatal. Contoh algoritma *lossless* adalah *Arithmetic Coding*, *Huffman Coding*, dan lain-lain.

2. Kompresi *Lossy*

Kompresi *lossy* adalah kompresi yang menekankan pada perubahan atau hilangnya beberapa informasi atau bit pada data, sehingga hasil kompresi tidak bisa lagi dikembalikan ke bentuk semula (*irreversible*). Namun, hasil kompresi masih bisa mempertahankan informasi utama pada data. Kompresi ini cocok diaplikasikan pada *file* suara, gambar atau video. Umumnya teknik ini menghasilkan kualitas hasil kompresi yang rendah, namun rasio kompresinya cenderung tinggi. Contoh algoritma kompresi *lossy* adalah *Fractal Compression*, *Wavelet Compression*, *Wyner-Ziv Coding* (WZC), dan lain-lain.

## 2.2 Burrows-Wheeler Transform (BWT)

Algoritma BWT merupakan algoritma proses melakukan transformasi terhadap blok data teks menjadi suatu bentuk baru yang tetap mengandung karakter yang sama, hanya saja urutannya yang berbeda.

Transformasi ini cenderung mengelompokan karakter secara berurut sehingga peluang untuk menemukan karakter yang sama secara berurutan akan meningkat sehingga akan lebih mudah dikompresi oleh algoritma kompresi seperti *Run-Length Encoding*, *Move-To-Front*, atau *Huffman Coding*. Menurut Burrows dan Wheeler [6], karena algoritma BWT hanya melakuan perubahan urutan karakter dari data sebelumnya, maka algoritma ini bersifat reversible.

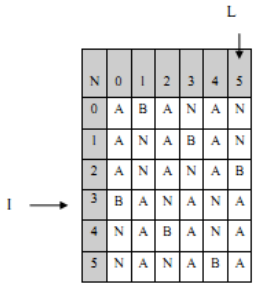
Sebagai contoh untuk melakukan transformasi *(encoding)* pada string "BANANA" dapat dilihat pada langkah-langkah berikut [4] :

1. Melakukan rotasi *(cyclic shift)* pada string "BANANA" sebanyak *N-1* kali, sehingga diperoleh matrik yang berordo *NxN* :

| N | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | B | A | N | A | N | A |
| 1 | A | N | A | N | A | B |
| 2 | N | A | N | A | B | A |
| 3 | A | N | A | B | A | N |
| 4 | N | A | B | A | N | A |
| 5 | A | B | A | N | A | N |

**Gambar 1. Matriks *String* "BANANA" [4]**

2. Mengurutkan matriks hasil rotasi secara *lexicographic* pada baris-baris matriks. :

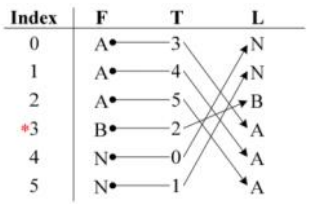



| N | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | A | B | A | N | A | N |
| 1 | A | N | A | B | A | N |
| 2 | A | N | A | N | A | B |
| 3 | B | A | N | A | N | A |
| 4 | N | A | B | A | N | A |
| 5 | N | A | N | A | B | A |

**Gambar 2. Matriks *String* "BANANA" Yang Diurutkan Secara *lexicographic* [4]**

Berdasarkan Gambar 2. tersebut diperoleh *string L* yang dibentuk dari karakter terakhir setiap baris matriks, dan *index I* yang menyatakan posisi string yang asli, sehingga hasil *encoding* adalah *(L, I)* yang dalam hal ini adalah (NNBAAA, 3).

*Decoding* pada algoritma BWT sedikit berbeda dengan *encoding*. Jika pada *encoding* sebuah matriks dibuat secara utuh, akan tetapi pada *decoding*, matriks yang dibuat hanya terdiri dari kolom pertama *F* dan kolom terakhir *L*. Dimana *F* diperoleh dengan cara mengurutkan *L* secara *lexicographic*.

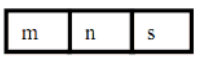



**Gambar 3. Transformasi Encoded String (NNBAAA, 3) [4]**

Karena index *I* dari *string* telah diketahui pada saat *encoding*, maka akan diketahui pula karakter pertama dan terakhir dari *string* pada *F* dan *L*. Sehingga dengan adanya informasi ini, maka *string* asli akan dapat diperoleh.

## 2.3 Run-Length Encoding (RLE)

*Run-Length Encoding* merupakan algoritma yang memanfaatkan karakter-karakter yang berulang secara berurutan pada data yaitu dengan mengkodekannya dengan sebuah *string* yang terdiri dari jumlah perulangan karakter yang terjadi, diikuti dengan sebuah karakter yang berulang tersebut [8]. Sehingga jumlah karakter yang berulang pada data merupakan penentu keberhasilan kompresi algoritma RLE. Secara garis besar format kode yang dihasilkan oleh algoritma RLE dituliskan sebagai berikut :




**Gambar 4. Format Kode Algoritma RLE [7]**

Keterangan gambar di atas yaitu, dimana *m* adalah penanda (*marker byte*), *n* adalah jumlah deret karakter yang berulang, dan *s* sebuah karakter yang berulang tersebut.

Sebagai contoh apabila string: "AAAABBCDEEEEEFGHHHIJ" dikompresi dengan algoritma RLE maka hasilnya adalah : "^4A2^B^1C^1D^5E^1F^1G^3H^1I^1J". Karena panjang

kode yang dihasilkan oleh RLE untuk setiap deret karakter minimal 3 *byte*, maka jumlah perulangan karakter harus lebih dari 3 (tiga) kali agar pengkodean bisa dilakukan. Satu hal yang perlu diperhatikan untuk penanda m adalah, sebaiknya yang dipilih adalah karakter yang jarang digunakan pada data (seperti tanda #, ^, |, atau ~).

Algoritma *decoding* pada RLE cukup sederhana yaitu mengembalikan karakter asli berdasarkan jumlah perulangan setiap karakter pada hasil kompresi. Proses *decoding* dilakukan langkah-langkah berikut ini :

1. Lihat karakter pada hasil pemampatan satu-persatu dari awal sampai akhir, jikaditemukan bit penanda, lakukan proses pengembalian.
2. Lihat karakter setelah bit penanda, konversikan ke bilangan desimal untuk menentukan jumlah karakter yang berurutan.
3. Lihat karakter berikutnya, kemudian lakukan penulisan karakter tersebut sebanyak bilangan yang telah diperoleh pada karakter sebelumnya (point 2).

Misalnya, apabila hasil kompresi adalah "^4A2^B^1C^1D^5E^1F^1G^3H^1I^1J" maka hasil dekompresi "AAAABBCDEEEEEFGHHHIJ".

Pada umumnya algoritma RLE optimal digunakan pada *file-file* yang memiliki karakter-karakter yang cenderung homogen. Oleh karena itu, jika algoritma tersebut dipergunakan secara *universal* maka perlu dilakukan pengelompokan atau transformasi karakter-karakter / simbol-simbol yang sejenis.

## 2.4 Move-To-Front Coding (RLE)

*Move-To-Front* (MTF) *coding* adalah teknik yang mengkodekan aliran simbol berdasarkan pada kode adaptasi. Simbol dikodekan oleh posisinya sendiri di daftar setiap simbol dalam alfabet. Awalnya daftar diurutkan secara *lexicographic* (atau cara lain yang ditentukan). Setelah simbol dalam aliran data dikodekan, simbol tersebut dipindahkan dari posisi semula ke depan daftar dan simbol terdepan dipindahkan satu posisi ke belakang [9].

Sebagai contoh diberikan daftar urutan alphabet sebagai acuan dalam pengkodean, yaitu **(' ','.','a','e','h','i','s','t')** dengan input **"ssat tt hiies ."** berikut proses *encoding* berlangsung :

**Tabel 1. Proses *Encoding* Algoritma *Move-To-Front* [4]**

| Renaming Symbols | Symbol List | Encoded Symbols |
|---|---|---|
| ssat tt hiies . | ' ','.','a','e','h','i','s','t' | |
| sat tt hiies . | 's',' ','.','a','e','h','i','t' | 6 |
| at tt hiies . | 's',' ','.','a','e','h','i','t' | 6,0 |
| t tt hiies . | 'a','s',' ','.','e','h','i','t' | 6,0,3 |
| tt hiies . | 't','a','s',' ','.','e','h','i' | 6,0,3,7 |
| tt hiies . | ' ','t','a','s','.','e','h','i' | 6,0,3,7,3 |
| t hiies . | 't',' ','a','s','.','e','h','i' | 6,0,3,7,3,1 |
| hiies . | 't',' ','a','s','.','e','h','i' | 6,0,3,7,3,1,0 |
| hiies . | ' ','t','a','s','.','e','h','i' | 6,0,3,7,3,1,0,1 |
| iies . | 'h',' ','t','a','s','.','e','i' | 6,0,3,7,3,1,0,1,6 |
| ies . | 'i','h',' ','t','a','s','.','e' | 6,0,3,7,3,1,0,1,6,7 |
| es . | 'i','h',' ','t','a','s','.','e' | 6,0,3,7,3,1,0,1,6,7,0 |
| s . | 'e','i','h',' ','t','a','s','.' | 6,0,3,7,3,1,0,1,6,7,0,7 |
| . | 's','e','i','h',' ','t','a','.' | 6,0,3,7,3,1,0,1,6,7,0,7,6 |
| . | ' ','s','e','i','h','t','a','.' | 6,0,3,7,3,1,0,1,6,7,0,7,6,4 |
| | '.',' ','s','e','i','h','t','a' | 6,0,3,7,3,1,0,1,6,7,0,7,6,4,7 |

Berbeda dengan proses *decoding* BWT, Proses *decoding Move-To-Front* (MTF) lebih singkat dan cukup jelas. Proses ini secara teknis seperti proses *encoding*, tapi lebih intensif memakan waktu [8]. Pada proses ini posisi sebuah simbol dalam daftar simbol setiap alfabet yang nantinya digunakan untuk memecahkan kode simbol. Seperti proses *encode*, Ketentuan daftar simbol dapat secara *lexicographic* (atau cara lain yang telah ditentukan). Data yang telah dikodekan menunjukkan posisi simbol diterjemahkan. Setelah simbol dilakukan *decoding*, simbol dipindahkan ke bagian depan daftar. Berikut ini tabel *decode* dari ketentuan sebelumya :

**Tabel 2. Proses *Decoding* Algoritma *Move-To-Front* [4]**

| Renaming Symbols | Symbol List | Decoded Symbols |
|---|---|---|
| 6,0,3,7,3,1,0,1,6,7,0,7,6,4,7 | ' ','.','a','e','h','i','s','t' | |
| 0,3,7,3,1,0,1,6,7,0,7,6,4,7 | 's',' ','.','a','e','h','i','t' | s |
| 3,7,3,1,0,1,6,7,0,7,6,4,7 | 's',' ','.','a','e','h','i','t' | ss |
| 7,3,1,0,1,6,7,0,7,6,4,7 | 'a','s',' ','.','e','h','i','t' | ssa |
| 3,1,0,1,6,7,0,7,6,4,7 | 't','a','s',' ','.','e','h','i' | ssat |
| 1,0,1,6,7,0,7,6,4,7 | ' ','t','a','s','.','e','h','i' | ssat_ |
| 0,1,6,7,0,7,6,4,7 | 't',' ','a','s','.','e','h','i' | ssat t |
| 1,6,7,0,7,6,4,7 | 't',' ','a','s','.','e','h','i' | ssat tt |
| 6,7,0,7,6,4,7 | ' ','t','a','s','.','e','h','i' | ssat tt_ |
| 7,0,7,6,4,7 | 'h',' ','t','a','s','.','e','i' | ssat tt h |
| 7,0,7,6,4,7 | 'h',' ','t','a','s','.','e','i' | ssat tt h |
| 0,7,6,4,7 | 'i','h',' ','t','a','s','.','e' | ssat tt hi |
| 7,6,4,7 | 'i','h',' ','t','a','s','.','e' | ssat tt hii |
| 6,4,7 | 'e','i','h',' ','t','a','s','.' | ssat tt hiie |
| 4,7 | 's','e','i','h',' ','t','a','.' | ssat tt hiies |
| 7 | ' ','s','e','i','h','t','a','.' | ssat tt hiies_ |
| | '.',' ','s','e','i','h','t','a' | ssat tt hiies . |

## 2.5 Huffman Coding

Algoritma *Huffman* adalah salah satu metode yang paling terkenal dalam kompresi teks, algoritma ini dibuat oleh *David Huffman* pada tahun 1952. Algoritma *Huffman* menggunakan prinsip pengkodean yang mirip dengan kode *Morse*, yaitu tiap karakter dikodekan hanya dengan rangkaian beberapa bit, dimana karakter yang sering muncul dikodekan dengan rangkaian bit yang pendek dan karakter yang jarang muncul dikodekan dengan rangkaian bit yang lebih panjang [10].

Algoritma kompresi *Huffman* menghasilkan kode bit yang lebih sedikit atau efisien untuk karakter-karakter yang sering muncul dalam data teks. Untuk membuktikan hal tersebut, maka dibuatlah pohon biner *Huffman*. Kode tersebut diperoleh dengan cara memyusun sebuah pohon biner *Huffman* untuk masing-masing simbul berdasarkan nilai probabilitasnya. Simpul yang memiliki probabilitas terbesar akan dekat dengan *root* dan simpul yang memiliki probabilitas terkecil akan terletak jauh dari *root* [5]. Langkah-langkah pembuatan pohon biner *Huffman* adalah :

1. Pengurutan keluaran sumber berdasarkan frekuensi kemunculan.
2. Menggabungkan dua keluaran yang memiliki frekuensi terendah.
3. Memberikan nilai kode bit 0 di sebelah kiri dan kode bit 1 di sebelah kanan.
4. Apabila sebuah keluaran merupakan hasil dari penggabungan dua keluaran, maka berikan kode bit 0 dan 1 untuk kode *word*-nya, lakukan proses ini hingga terbentuk akar.

Berikut ini adalah contoh penerapan algoritma kompresi *Huffman*. Sebuah *file* berisi karakter "MATA-MATA" dimana

jika diuraikan berdasarkan kode ASCII, maka akan terlihat sebagai berikut :

**Tabel 3. Konversi Karakter dalam Kode Biner ASCII [10]**

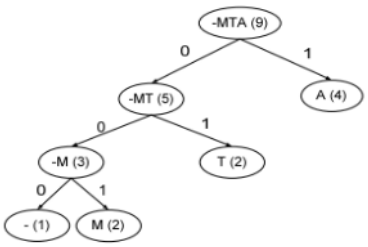
| KARAKTER | KODE BINER ASCII |
|---|---|
| M | 0100 1101 |
| A | 0100 0001 |
| - | 0010 1101 |
| T | 0101 0100 |

Setelah membaca setiap karakter yang ada dalam teks, kemudian dihitung frekuensi kemunculannya.

**Tabel 4. Frekuensi Kemunculan Karakter [10]**

| KARAKTER | FREKUENSI |
|---|---|
| A | 4/9 |
| M | 2/9 |
| T | 2/9 |
| - | 1/9 |

Penghitungan selesai, langkah berikutnya adalah membuat pohon Huffman.




**Gambar 5. Pohon algoritma *Huffman* [10]**

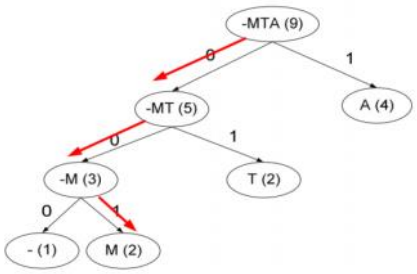
Didapat kode baru untuk masing-masing karakter adalah sebagai berikut :

**Tabel 5. Kode Bit Hasil Kompresi [10]**

| KARAKTER | KODE BINER |
|---|---|
| A | 1 |
| M | 001 |
| T | 01 |
| - | 000 |

Dekompresi teks dapat dilakukan dengan dua cara, yang pertama dengan menggunakan pohon biner *Huffman* dan yang kedua dengan menggunakan tabel kode biner *Huffman*. Langkah-langkah mengdekompresi suatu kode biner yang merupakan hasil dari proses kompresi dengan menggunakan pohon biner *Huffman* adalah sebagai berikut :

1. Baca sebuah bit dari kode biner.
2. Menelusuri pohon mulai dari atas atau akar, periksa ke kanan dan kekiri.
3. Ulangi langkah 1 dan 2 sampai bertemu daun. Kodekan rangkaian bit yang telah dibaca dengan karakter daun.
4. Ulangi dari langkah 1 sampai semua bit di dalam kode biner terbaca semua dan berubah menjadi karakter-karakter yang sesuai pada daun.



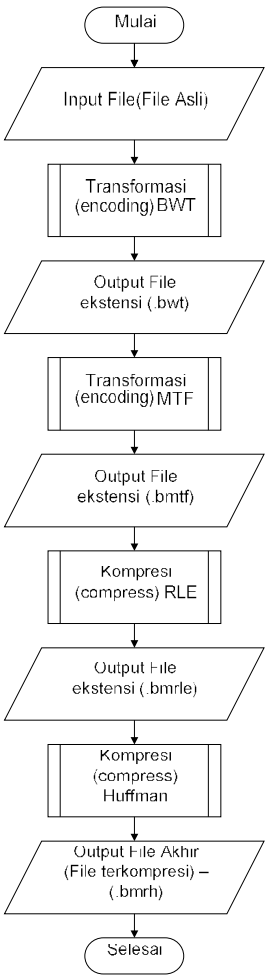

**Gambar 6. Proses Dekompresi dengan pohon algoritma *Huffman* [10]**

## 3. METODE PENELITIAN

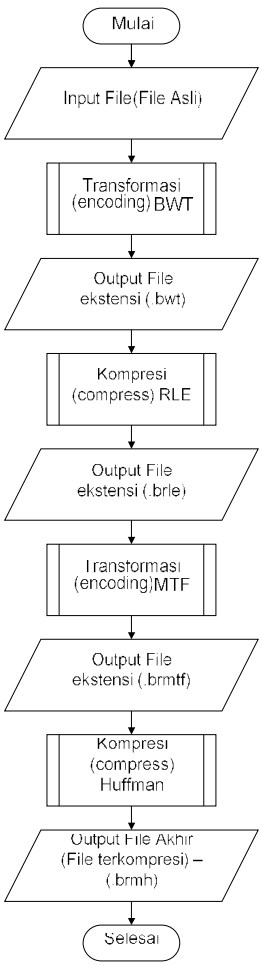
Metode dan teknik yang digunakan dalam implementasi kasus *data compression* ini terbagi dalam 3 tahap, yaitu tahap kompresi, tahap dekompresi, dan tahap pengujian dan perbandingan kinerja dua kombinasi algoritma. Berikut gambaran secara keseluruhan alur penyelesaian :

### 3.1 Tahap Kompresi

Gambar 7 dan gambar 8 berikut menunjukkan diagram alir (*flowchart*) rancangan proses *encoding* (kompresi) kombinasi penggabungan algoritma BWT, RLE, MTF da n *Huffman*.




**Gambar 7. *Flowchart* Kompresi BMRH**



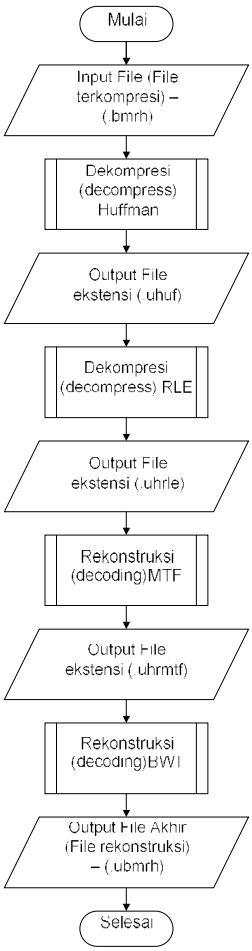

**Gambar 8. *Flowchart* Kompresi BRMH**

Tahap kompresi ini, *file* dikompresi dengan menggunakan kombinasi algoritma BMRH dan BRMH kemudian akan menghasilkan sebuah output file kompresi yang berekstensi (.bmrh) untuk kombinasi BMRH dan (.brmh) untuk kombinasi BRMH. Metode - metode kompresi yang digunakan terdiri dari 2 algoritma transformasi (BWT dan MTF) dan 2 algoritma kompresi (RLE dan Huffman) dengan perbedaan hanya pada komposisi.

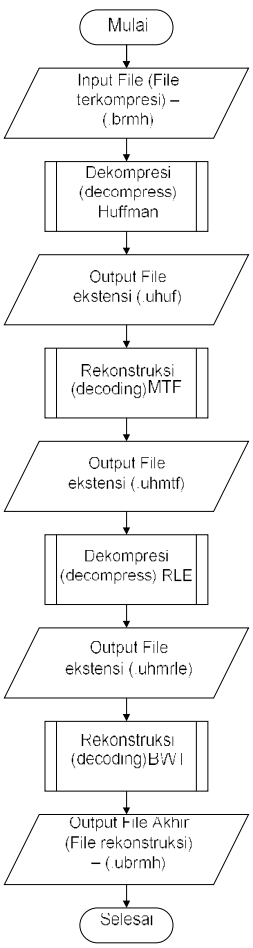
## 3.2 Tahap Dekompresi

Gambar 9 dan gambar 10 berikut menunjukkan diagram alir (*flowchart*) rancangan proses d*ecoding* (dekompresi) kombinasi penggabungan algoritma BWT, RLE, MTF dan *Huffman* yang berupa proses kebalikan dari tahap kompresi.

Tahap dekompresi ini, *file* didekompresi dengan menggunakan kombinasi algoritma BMRH dan BRMH sehingga akan menghasilkan sebuah output file rekonstruksi yang berekstensi (.ubmrh) untuk kombinasi BMRH dan (.ubrmh) untuk kombinasi BRMH.




**Gambar 9. *Flowchart* Dekompresi BMRH**




**Gambar 10. *Flowchart* Dekompresi BRMH**

## 3.3 Tahap Pengujian dan Perbandingan

Pada tahap pengujian dan perbandingan dilakukan perhitungan kinerja dengan beberapa ketentuan dan parameter pengujian sebagai tolok ukur efektivitas algoritma serta beberapa jenis *file* uji yang sudah ditentukan dan pada batasan masalah, seperti berikut ini :

1. Pengujian kinerja tiap algoritma pada *file* uji

Perhitungan performansi dengan *rasio performansi kompresi* dengan rumus sebagai berikut [11] :

$$Rasio = 1 - \frac{\text{Ukuran File kompresi } (byte)}{\text{Ukuran File Asli } (byte)} \times 100\% \quad ..1$$

Perhitungan rentang *waktu kompresi* dan *dekompresi* per satuan *byte* agar dapat dibandingkan pada perhitungan *waktu kompresi* dan *dekompresi* untuk tiap ukuran *file* dengan rumus sebagai berikut [11] :

$$Waktu\ Kompresi\ per\ Byte = \frac{\text{Waktu Kompresi } (second)}{\text{Ukuran File Asli } (byte)} \quad ..2$$

$$Waktu\ Dekompresi\ per\ Byte = \frac{\text{Waktu Dekompresi } (second)}{\text{Ukuran File Kompresi } (byte)} \quad ..3$$

*File-file* yang akan diujikan untuk proses kompresi dan dekompresi menggunakan format *Text* (.txt), *Rich Text Format* (.rtf), *Document* (.doc), *Application* (.exe), *Bitmap* (.bmp), *Tagged Image Format* (.tif) dan *Application Extension / Library* (.dll). Pengujian dilakukan sebanyak 420 kali meliputi uji kompresi maupun dekompresi pada kedua algoritma (BMRH dan BRMH), dengan menggunakan 105 *file* yang terdiri dari beberapa ukuran *file*.

2. Membandingkan kinerja algoritma yang diuji

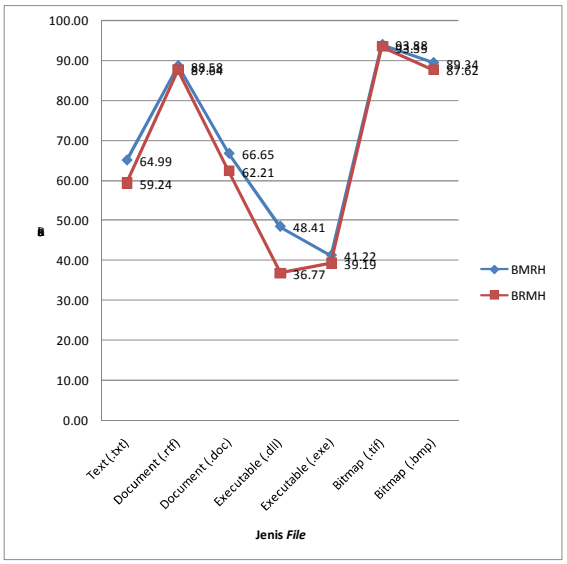
Setelah pengujian kinerja, dilakukan perbandingan hasil berdasarkan parameter waktu dan rasio kompresi yang telah dihitung pada tahap pengujian. Kemudian menarik kesimpulan dari hasil perbandingan tersebut.

# 4. HASIL DAN PEMBAHASAN

Pada hasil dan pembahasan dilakukan perbandingan kinerja kombinasi algoritma BMRH dan BRMH berdasarkan variabel terkait yang ditentukan seperti rasio dan waktu kompresi serta waktu dekompresi. Hasil disajikan dalam bentuk diagram kemudian dilakukan analisis perbandingan baik dalam taraf jenis *file* maupun rata-rata total keseluruhan.

## 4.1 Perbandingan Rasio Kompresi

Tabel 6 dan Gambar 11 di bawah ini menunjukkan perbandingan rasio kompresi antara kombinasi algoritma BMRH dan BRMH yang ditinjau dari tiap jenis *file*.




**Gambar 11. Grafik Hasil Perbandingan Rasio Algoritma BMRH dan BRMH**

**Tabel 6. Hasil Pengujian Rasio Algoritma BMRH dan BRMH**

| Jenis *File* | Rasio (%) | |
|---|---|---|
| | BMRH | BRMH |
| Text (.txt) 0 - 1 MB | 67.73 | 65.43 |
| Text (.txt) 1 - 3 MB | 70.90 | 64.88 |
| Text (.txt) 3 - 6 MB | 56.35 | 47.41 |
| **Rata - Rata Text (.txt)** | **64.99** | **59.24** |
| Document (.rtf) 0 - 1 MB | 86.02 | 85.09 |
| Document (.rtf) 1 - 3 MB | 89.72 | 89.20 |
| Document (.rtf) 3 - 6 MB | 90.01 | 88.61 |
| **Rata - Rata Document (.rtf)** | **88.58** | **87.64** |
| Document (.doc) 0 - 1 MB | 66.92 | 64.12 |
| Document (.doc) 1 - 3 MB | 69.69 | 66.92 |
| Document (.doc) 3 - 6 MB | 63.35 | 55.60 |
| **Rata - Rata Document (.doc)** | **66.65** | **62.21** |
| Executable (.dll) 0 - 1 MB | 47.82 | 33.03 |
| Executable (.dll) 1 - 3 MB | 53.15 | 49.75 |
| Executable (.dll) 3 - 6 MB | 44.26 | 27.52 |
| **Rata - Rata Executable (.dll)** | **48.41** | **36.77** |
| Executable (.exe) 0 - 1 MB | 47.31 | 46.54 |
| Executable (.exe) 2 - 3 MB | 37.34 | 30.22 |
| Executable (.exe) 3 - 6 MB | 39.02 | 40.80 |
| **Rata - Rata Executable (.exe)** | **41.22** | **39.19** |
| Bitmap (.tif) 0 - 1 MB | 92.62 | 92.54 |
| Bitmap (.tif) 1 - 3 MB | 95.18 | 94.91 |
| Bitmap (.tif) 3 - 6 MB | 93.84 | 92.59 |
| **Rata - Rata Bitmap (.tif)** | **93.88** | **93.35** |
| Bitmap (.bmp) 0 - 1 MB | 84.43 | 79.51 |
| Bitmap (.bmp) 2 - 3 MB | 94.15 | 93.62 |
| Bitmap (.bmp) 3 - 6 MB | 89.42 | 89.73 |
| **Rata - Rata Bitmap (.bmp)** | **89.34** | **87.62** |
| **Rata - Rata Total** | **70.44** | **66.57** |

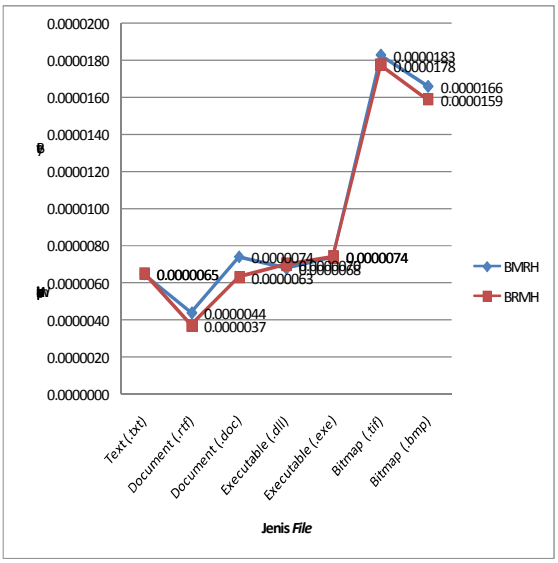
Berdasarkan hasil perbandingan rasio kompresi yang ditampilkan lewat Tabel 6 dan Gambar 11. Terlihat bahwa *file (.rtf), (.tif),* dan *(.bmp)* memiliki rasio kompresi yang paling signifikan (hampir mendekati 90 %) diantara keempat jenis *file* lainnya yaitu, berturut-turut 88.58 %, 93.88 % dan 89.34 % untuk kominasi BMRH serta 87.64 %, 93.35 % dan 87.62 % untuk kombinasi BRMH. Bila ditinjau dari rata-rata total algoritma BMRH memberikan rasio kompresi yang lebih tinggi 3.87 % dibanding algoritma BRMH begitu pula bila ditinjau tiap jenis *file*. Secara keseluruhan algoritma BMRH lebih efektif dalam kompresi dibanding algoritma BRMH.

## 4.2 Perbandingan Waktu Kompresi

Tabel 7 dan Gambar 12 di bawah ini menunjukkan perbandingan waktu kompresi per *byte* antara kombinasi algoritma BMRH dan BRMH yang ditinjau dari tiap jenis *file*.

**Tabel 7. Hasil Pengujian Waktu Kompresi Algoritma BMRH dan BRMH**

| Jenis *File* | Waktu Kompresi per *Byte* (*second*) | |
|---|---|---|
| | BMRH | BRMH |
| Text (.txt) 0 - 1 MB | 0.0000036 | 0.0000040 |
| Text (.txt) 1 - 3 MB | 0.0000109 | 0.0000110 |
| Text (.txt) 3 - 6 MB | 0.0000050 | 0.0000044 |
| **Rata - Rata Text (.txt)** | **0.0000065** | **0.0000065** |
| Document (.rtf) 0 - 1 MB | 0.0000049 | 0.0000041 |
| Document (.rtf) 1 - 3 MB | 0.0000041 | 0.0000035 |
| Document (.rtf) 3 - 6 MB | 0.0000043 | 0.0000035 |
| **Rata - Rata Document (.rtf)** | **0.0000044** | **0.0000037** |
| Document (.doc) 0 - 1 MB | 0.0000068 | 0.0000058 |
| Document (.doc) 1 - 3 MB | 0.0000065 | 0.0000054 |
| Document (.doc) 3 - 6 MB | 0.0000089 | 0.0000076 |
| **Rata - Rata Document (.doc)** | **0.0000074** | **0.0000063** |
| Executable (.dll) 0 - 1 MB | 0.0000054 | 0.0000067 |
| Executable (.dll) 1 - 3 MB | 0.0000077 | 0.0000075 |
| Executable (.dll) 3 - 6 MB | 0.0000073 | 0.0000067 |
| **Rata - Rata Executable (.dll)** | **0.0000068** | **0.0000070** |
| Executable (.exe) 0 - 1 MB | 0.0000064 | 0.0000069 |
| Executable (.exe) 2 - 3 MB | 0.0000099 | 0.0000097 |
| Executable (.exe) 3 - 6 MB | 0.0000058 | 0.0000055 |
| **Rata - Rata Executable (.exe)** | **0.0000074** | **0.0000074** |
| Bitmap (.tif) 0 - 1 MB | 0.0000046 | 0.0000039 |
| Bitmap (.tif) 1 - 3 MB | 0.0000066 | 0.0000062 |
| Bitmap (.tif) 3 - 6 MB | 0.0000437 | 0.0000431 |
| **Rata - Rata Bitmap (.tif)** | **0.0000183** | **0.0000178** |
| Bitmap (.bmp) 0 - 1 MB | 0.0000056 | 0.0000036 |
| Bitmap (.bmp) 2 - 3 MB | 0.0000034 | 0.0000034 |
| Bitmap (.bmp) 3 - 6 MB | 0.0000410 | 0.0000408 |
| **Rata - Rata Bitmap (.bmp)** | **0.0000166** | **0.0000159** |
| **Rata - Rata Total** | **0.0000096** | **0.0000092** |

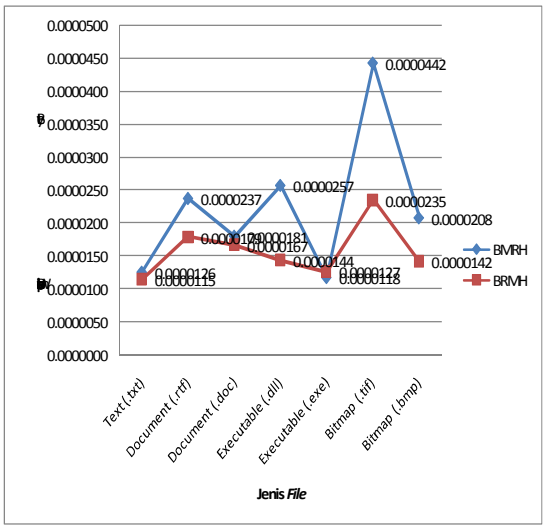



**Gambar 12. Grafik Hasil Perbandingan Waktu Kompresi Algoritma BMRH dan BRMH**

Berdasarkan hasil perbandingan waktu kompresi yang ditampilkan lewat Tabel 7. dan Gambar 12, hasil perbandingan waktu kompresi algoritma BMRH dan BRMH menunjukkan tidak tampak adanya perbedaan yang cukup berarti diantara kedua algoritma ditinjau dari waktu kompresi dilihat dari bentuk grafik antara BMRH dan BRMH bahkan pada *file (.exe)* dan *(.txt)* memiliki rata-rata waktu kompresi yang sama. *File (.rtf)* pada pengujian ini memiliki rata-rata waktu kompresi yang paling rendah diantara jenis *file* uji lainnya. Sedangkan *file (.tif)* dan *(.bmp)* memiliki interval perbedaan waktu kompresi yang paling tinggi dengan *file* lainnya. Pada perhitungan variabel ini terlihat jelas bahwa rata-rata total waktu kompresi baik kombinasi BMRH dan BRMH memiliki perbedaan yang tidak terlalu signifikan.

## 4.3 Perbandingan Waktu Dekompresi

Tabel 8 dan Gambar 13 di bawah ini menunjukkan perbandingan waktu dekompresi per *byte* antara kombinasi algoritma BMRH dan BRMH yang ditinjau dari tiap jenis *file*.




**Gambar 13. Grafik Hasil Perbandingan Waktu Dekompresi Algoritma BMRH dan BRMH**

**Tabel 8. Hasil Pengujian Waktu Dekompresi Algoritma BMRH dan BRMH**

| Jenis *File* | Waktu Dekompresi per *Byte* (*second*) | |
|---|---|---|
| | **BMRH** | **BRMH** |
| Text (.txt) 0 - 1 MB | 0.0000130 | 0.0000115 |
| Text (.txt) 1 - 3 MB | 0.0000126 | 0.0000116 |
| Text (.txt) 3 - 6 MB | 0.0000123 | 0.0000114 |
| **Rata - Rata Text (.txt)** | **0.0000126** | **0.0000115** |
| Document (.rtf) 0 - 1 MB | 0.0000237 | 0.0000170 |
| Document (.rtf) 1 - 3 MB | 0.0000239 | 0.0000182 |
| Document (.rtf) 3 - 6 MB | 0.0000237 | 0.0000185 |
| **Rata - Rata Document (.rtf)** | **0.0000237** | **0.0000179** |
| Document (.doc) 0 - 1 MB | 0.0000182 | 0.0000168 |
| Document (.doc) 1 - 3 MB | 0.0000186 | 0.0000169 |
| Document (.doc) 3 - 6 MB | 0.0000173 | 0.0000164 |
| **Rata - Rata Document (.doc)** | **0.0000181** | **0.0000167** |
| Executable (.dll) 0 - 1 MB | 0.0000132 | 0.0000147 |
| Executable (.dll) 1 - 3 MB | 0.0000161 | 0.0000150 |
| Executable (.dll) 3 - 6 MB | 0.0000479 | 0.0000135 |
| **Rata - Rata Executable (.dll)** | **0.0000257** | **0.0000144** |
| Executable (.exe) 0 - 1 MB | 0.0000117 | 0.0000153 |
| Executable (.exe) 2 - 3 MB | 0.0000130 | 0.0000122 |
| Executable (.exe) 3 - 6 MB | 0.0000107 | 0.0000107 |
| **Rata - Rata Executable (.exe)** | **0.0000118** | **0.0000127** |
| Bitmap (.tif) 0 - 1 MB | 0.0000334 | 0.0000205 |
| Bitmap (.tif) 1 - 3 MB | 0.0000477 | 0.0000275 |
| Bitmap (.tif) 3 - 6 MB | 0.0000516 | 0.0000224 |
| **Rata - Rata Bitmap (.tif)** | **0.0000442** | **0.0000235** |
| Bitmap (.bmp) 0 - 1 MB | 0.0000183 | 0.0000134 |
| Bitmap (.bmp) 2 - 3 MB | 0.0000218 | 0.0000147 |
| Bitmap (.bmp) 3 - 6 MB | 0.0000225 | 0.0000145 |
| **Rata - Rata Bitmap (.bmp)** | **0.0000442** | **0.0000235** |
| **Rata - Rata Total** | **0.0000224** | **0.0000158** |

Berdasarkan hasil perbandingan waktu dekompresi yang ditampilkan lewat Tabel 8 dan Gambar 13, Apabila ditinjau dari gambar grafik, didapat bahwa kombinasi algoritma BRMH unggul (waktu dekompresi lebih rendah) pada pengujian *file (.txt), (.rtf), (.doc), (.dll), (.tif),* dan *(.bmp)*. Sedangkan untuk *file (.exe)* lebih unggul kombinasi algoritma BMRH dengan interval perbedaan tidak terlalu besar dengan BRMH. Jika dilihat dari rata-rata total, secara keseluruhan algoritma BRMH lebih cepat dalam dekompresi dibanding algoritma BMRH.

Hasil kompresi terhadap beberapa *file* uji didapat bahwa rasio kompresi kombinasi algoritma BMRH lebih besar dibanding kombinasi BRMH dilihat dari presentase reduksi yang lebih besar. Hal tersebut terjadi karena ada beberapa perilaku yang perlu diperhatikan dalam penggabungan algoritma *Run-Length Encoding* (RLE) dan *Move-To-Front* (MTF) dalam kompresi kombinasi BRMH.

Pada kombinasi BRMH (BWT – RLE – MTF – *Huffman*), proses kombinasi tahap penggabungan metode kedua, yaitu BWT – RLE secara perilaku proses kompresi berjalan sesuai dengan pemaparan Burrows & Wheeler (1994) yang dapat meningkatkan kinerja RLE menjadi lebih kompresif karena sifat algoritma BWT pada dasarnya mengumpulkan *byte* / karakter sejenis dalam suatu *source file* yang kemudian dapat dikodekan

dengan mudah oleh RLE dengan mengganti kelompok karakter sejenis tersebut menjadi 3 karakter saja (*marker byte*, *byte* indeks numerik, dan karakter itu sendiri). Akan tetapi ketika mencapai tahap penggabungan metode ketiga yaitu BWT – RLE – MTF, operasi *encoding* MTF akan memindahkan *byte* indeks numerik baru tersebut ke depan karena terdeteksi sebagai varian simbol yang baru dan akhirnya memungkinkan bertambahnya *byte* – *byte* dari simbol yang baru. Hingga akhirnya pada tahap penggabungan metode yang terakhir yaitu BWT – RLE – MTF – *Huffman*, metode Huffman melakukan kompresi berdasar frekuensi kemunculan karakter, dan tiap jenis karakter dilakukan *bit-level encoding* berdasarkan *mapping* pohon *Huffman*. Keberhasilan dari metode *Huffman* ini sangat bergantung pada metode sebelumnya. Semakin sedikit varian *byte* / karakter yang dihasilkan maka output dari metode *Huffman* yang dihasilkan akan lebih kecil ukurannya.

Sedangkan kombinasi BMRH (BWT – MTF – RLE – *Huffman*), proses yang dijalankan hingga tahap penggabungan metode kedua yaitu BWT – MTF tidak bersifat kompresif tetapi hasil yang didapat berpengaruh pada proses kompresi selanjutnya. Output BWT yang dikenai MTF dikodekan menjadi karakter numerik. Sebelumnya algoritma BWT telah mengelompokkan karakter sejenis maka MTF akan melakukan *encoding* pada saat pergantian karakter saja, selebihnya karakter sejenis yang mengikuti akan dikodekan dengan numerik "0" hingga bertemu kembali dengan varian karakter yang berbeda. Dengan demikian varian karakter menjadi lebih sedikit dan apabila terdapat banyak karakter sejenis dalam suatu *file* input maka frekuensi kemunculan karakter numerik "0" juga akan lebih banyak. Proses kompresi RLE pada tahap penggabungan metode ketiga, yaitu BWT – MTF – RLE tentunya akan lebih kompresif dikarenakan hasil output MTF yang memiliki varian karakter yang lebih sedikit serta frekuensi kemunculan karakter sejenis yang bertambah. Hingga akhirnya dengan metode *Huffman* pada tahap penggabungan yang terakhir BWT – MTF – RLE – *Huffman* terjadi proses reduksi ukuran *file* dengan mengkodekan karakter pada tingkat *bit-level* seperti penjelasan kombinasi BRMH sebelumnya.

Berdasarkan penjelasan tersebut dapat dikatakan bahwa kombinasi BMRH (BWT – MTF – RLE – *Huffman*) lebih unggul dalam rasio kompresi dikarenakan penggabungan BWT – MTF mampu menekan variasi karakter yang muncul serta meningkatkan kemunculan frekuensi karakter sejenis sehingga nantinya dalam proses kompresi baik RLE dan *Huffman* reduksi ukuran *file* dapat lebih besar. Sedangkan kombinasi untuk BRMH (BWT – RLE – MTF – *Huffman*), walaupun dalam penelitian sebelumnya penggabungan BWT – RLE (Telaumbanua, 2011) dan MTF – *Huffman* (Bentley, *et al*, 1986) mampu mereduksi ukuran *file*. Akan tetapi jika keduanya digabungkan maka pada perlakuan algoritma MTF dalam *encoding* yang sebelumnya dikenai algoritma BWT – RLE menyebabkan kurang maksimalnya dalam kompresi *file* karena akan cenderung memungkinkan menghasilkan *bytes* / karakter baru yang dapat menambah ukuran atau *size* objek kompresi.

## 5. KESIMPULAN

Setelah dilakukan pengujian dan perbandingan antara kedua kombinasi algoritma BMRH dan BRMH didapat besarnya rata-rata rasio kompresi dari terbesar hingga terkecil berturut-turut untuk pengujian kombinasi BMRH adalah *file* (.tif), (.bmp), (.rtf), (.doc), (.txt), (.dll), dan (exe) serta untuk pengujian kombinasi BRMH adalah *file* (.tif), (.rtf), (.bmp), (.doc), (.txt), (.exe), dan (.dll) yang secara keseluruhan unggul pada algoritma BMRH. Waktu yang dibutuhkan untuk proses kompresi dan dekompresi pada kedua kombinasi menunjukkan selisih perbedaan, untuk waktu kompresi BRMH lebih cepat $0.4 \times 10^{-6}$ *second/byte* dibanding BMRH, sedangkan untuk waktu dekompresi BRMH lebih cepat $6.6 \times 10^{-6}$ *second/byte* dibanding BMRH. Berdasarkan rata-rata total dari parameter-parameter yang diujikan didapat untuk kombinasi BMRH memiliki rasio kompresi 70.44 %, waktu kompresi $9.6 \times 10^{-6}$ *second/byte*, dan waktu dekompresi $22.4 \times 10^{-6}$ *second/byte*, sedangkan untuk kombinasi BRMH memiliki rasio kompresi 66.57 %, waktu kompresi $9.2 \times 10^{-6}$ *second/byte*, dan waktu dekompresi $15.8 \times 10^{-6}$ *second/byte*. Berdasarkan hasil rata-rata total yang didapat dapat disimpulkan BMRH lebih unggul dalam performansi kompresi data dibanding kombinasi BRMH.

## 6. SARAN

Adapun beberapa saran yang dapat diberikan untuk penelitian-penelitian selanjutnya yaitu mungkin kombinasi dapat dilakukan variasi dengan algoritma lain terutama algoritma *Huffman* dapat diganti dengan algoritma *statistical encoder* yang lain misal, *arithmetic coding*, *adaptive Huffman* atau *Shannon-Fano Coding*. Oleh karena waktu kompresi dan dekompresi yang masih terlihat cukup lama, maka dapat dilakukan modifikasi kombinasi penggabungan algoritma BRMH dan BMRH menjadi satu kesatuan algoritma kompresi baru supaya waktu eksekusi dapat lebih cepat dan permasalahan perilaku penggabungan satu sama lainnya dalam proses kompresi tidak terjadi.

## 7. REFERENSI


[1] Al-laham, M & Emary, M M E. 2007. Comparative Study Between Various Algorithms of Data Compression Techniques. *Proceeding of the World Congress on Engineering and Computer Science 2007*. San Francisco, USA. ISBN: 978-988-98671-6-4.

[2] Sharma, Mamta. 2010. Compression Using Huffman Coding. *IJCSNS International Journal of Computer Science and Network Security*, Vol. 10 No.5, May 2010.

[3] Shanmugasundaram, S & Lourdusamy, R. 2011. A Comparative Study Of Text Compression Algorithms. *International Journal of Wisdom Based Computing*, Vol. 1 (3), Desember 2011.

[4] Dipperstein, Michael. 2010. *Burrows-Wheeler Transform Discussion and Implementation*, 2000-2010: *Michael Dipperstein O'Stuff*, (Online), (http://michael.dipperstein.com/bwt/, diakses 06 Maret 2012 pukul 08.17 WIB).

[5] Dipperstein, Michael. 2010. *Huffman Code Discussion and Implementation*, 2000-2010: *Michael Dipperstein's Page O'Stuff*, (Online), (http://michael.dipperstein.com/huffman/, diakses 06 Maret 2012 pukul 08.18 WIB).

[6] Burrows, M & Wheeler, D. 1994. *A Block-sorting Lossless Data Compression Algorithm*. Technical Report 124, Digital Systems Research Center, Palo Alto, California, May 1994.

[7] Telaumbanua, Plipus. 2011. *Analisis Perbandingan Algoritma Kompresi Lempel Ziv Welch, Arithmetic Coding, Dan Run-Length Encoding Pada File Teks*. Publikasi Skripsi. Medan : Universitas Sumatera Utara.

[8] Dipperstein, Michael. 2010. *Run-Length Encoding (RLE) Discussion and Implementation*, 2000-2010: *Michael Dipperstein O'Stuff*, (Online), (http://michael.dipperstein.com/rle/, diakses 06 Maret 2012 pukul 08.15 WIB).

[9] Langer, Michael. 2008. *Move-to-front Algorithm*. http://www.cim.mcgill.ca/~langer/423/lecture8.pdf. Diakses Tanggal 06 Maret 2012 Pukul 08.09 WIB.

[10] Septin, Saputri. 2010. *Perbandingan Rasio Kompresi dan Efisiensi Algoritma Shannon-Fano dengan Algoritma Huffman Pada Kompresi Teks.* Publikasi Skripsi. Medan : Universitas Sumatera Utara.
[11] Pu Ida Mengyi. 2006. *Fundamental Data Compression*. London: Butterworth-Heinemann.