# *SIMULASI KOMPRESI DATA BZIP2 DENGAN MAPLE DAN APLIKASI INTERAKTIF MAPLET*

# *SIMULATING BZIP2 COMPRESSION USING MAPLE AND MAPLET INTERACTIVE APPLICATIONS*

*Hasniati, Loeky Haryanto, Armin Lawi*

*Jurusan Matematika, Fakultas MIPA, Universitas Hasanuddin*

**Alamat korespondensi:**

Hasniati
Jurusan Matematika
Fakultas MIPA
Universitas Hasanuddin
Makassar, 90245
HP : 085255804226
*Email : hasniati@kharisma.ac.id*

## Abstrak

Kompresi data adalah cara untuk mengurangi biaya penyimpanan dengan menghilangkan redundansi yang terjadi di sebagian besar file. Penelitian ini bertujuan memberikan simulasi salah satu versi kompresi lossless yang cukup populer untuk kompresi teks yaitu BZip2. Simulasi dibuat dengan menggunakan Maple dan tampilan interaktif Maplet sebagai implementasi dari merumuskan proses kompresi data terhadap teks. Proses kompresi melalui beberapa tahap transformasi dengan mengombinasi algoritma yang diusulkan yang dapat diklasifikasikan sebagai algoritma transformasi dan algoritma kompresi. Kompresi data teks Bzip2 menggunakan dua algoritma yaitu *BWT* dan *MTF* secara berurutan dan dilanjutkan dengan *Huffman encoding*. Metode yang digunakan dalam penelitian ini adalah: kajian literatur, pembuatan kode untuk progam dalam Maple dan Maplet, uji coba kode ke dalam program yang dilanjutkan dengan penulisan hasil akhir berdasarkan program yang dihasilkan. Hasil utama yang diperoleh dari tugas akhir ini adalah program Maple dan tampilan interaktif Maplet untuk menyajikan simulasi kompresi BZip2.

Kata Kunci: Transformasi Burrows-Wheeler (BW), transformasi Move-To-Front (MTF) dan pengkodean Huffman.

## Abstract

*Data compression is a way to reduce storage cost by eliminating redundancies that happen in most files. This paper aims to provide simulate a versions of lossless compression are quite popular for text is Bzip2. Simulations made using Maple and interactive display Maplet as the implementation to formulating text compression. Compression process through several steps of transformation by combining the proposed algorithm which can be classified as the transformation algorithms and compression algorithms. Bzip2 compression of text data using two algorithms are BWT and MTF, consecutively and followed by Huffman encoding. Methods used in this paper are: literature study, creating codes in Maple program and Maplet, testing the Maple program and Maplet applications using the codes and followed by writing a report based on the resulting outputs of the programs. The main results of this thesis is the Maple codes and Maplet interactive applications for simulating the BZip2 text compression*.

*Keywords: Data Compression, Burrows-Wheeler Tranformation* (*BWT*), *Move-To-Front Tranformation* (*MTF*), *Huffman Coding*

## PENDAHULUAN

Kompresi data adalah cara untuk mengurangi biaya penyimpanan dengan menghilangkan redundansi yang terjadi di sebagian besar file. Ada dua jenis kompresi yaitu *lossy* dan *lossless*. Kompresi *lossy* mengurangi ukuran file dengan menghilangkan beberapa data yang tidak dibutuhkan yang tidak akan dikenali oleh manusia setelah decoding, ini sering digunakan oleh kompresi video dan audio. Kompresi *lossless* di sisi lain, memanipulasi setiap bit data di dalam file untuk meminimalkan ukuran tanpa kehilangan data apapun setelah *decoding* (Suarjaya, 2012). Menurut Deorowic (2003), metode kompresi lossy dapat mencapai rasio kompresi yang lebih baik dari yang lossless. Sifat lossless dari kompresi teks sangat penting karena perbedaan yang sangat kecil antara teks asli dengan teks hasil rekonstruksi bisa memberikan makna yang sangat berbeda (Sayood, 2006).

Ada beberapa algoritma kompresi data yang populer. Dalam penelitian ini akan dibuat simulasi dari suatu proses kompresi data lossless yang cukup populer untuk kompresi teks yaitu BZip2, dimana proses ini melalui beberapa tahap transformasi dengan mengombinasi algoritma yang diusulkan (Solomon, 2007). Algoritma ini dapat diklasifikasikan sebagai algoritma transformasi dan algoritma kompresi. Pada implementasi Bzip2 digunakan tiga tahap yaitu: 1. transformasi *Burrows-Wheeler* (*BWT*) (Burrows, 1994), 2. transformasi Move-To-Front (*MTF*) dan 3. *arithmetic coding* atau *Huffman coding*, yang merupakan proses kompresi sesungguhnya (Munir, 2010). Proses ini diketahui sebagai block-shorting yang mana merupakan archiver data teks terbaik ditinjau dari kecepatan dan rasio kompresi yang dikemukakan Bastys (2010) dan Khalid (2006) dalam tulisannya.

Adapun efisiensi dari algoritma diatas diukur dari jumlah bit yang dihasilkan dari tiap-tiap kompresi data yang dihitung dengan menggunakan entropi data, dimana suatu data dikatakan optimum jika panjang rata-rata bit dari suatu sumber data sama dengan jumlah entropi dari data tersebut (Pu, 2006). Kumar (2012) dalam postingannya mengatakan bahwa *BWT* digunakan dalam produk *Avadis NGS*.

Penulisan jurnal ini bertujuan untuk menerapkan kompresi data teks dengan tiga tahapan pada untaian tertentu yaitu: transformasi *Burrows-Wheeler* (*BWT*), transformasi *Move-To-Front* (*MTF*) dan *arithmetic coding* atau *Huffman coding*. Jadi akan dibahas algoritma dan transformasi yang digunakan dalam tahapan-tahapan pada proses kompresi data BZip2. Karena belum ada simulasi dan contoh hasil proses kompresi dan dekompresi, maka di dalam penelitian ini akan dicoba mendapatkan hasil kompresi tiga tahap ini melalui simulasi dengan Maple.

## METODE PENELITIAN

### *Lokasi dan Rancangan Penelitian*

Penelitian ini bertempat di Jurusan Matematika FMIPA Universitas Hasanuddin. Rancangan penelitian ini berbentuk penelitian kualitatif dengan melakukan studi kepustakaan, dengan mengumpulkan bahan penelitian melalui penelusuran jurnal, literatur dan penelitian terkait masalah kompresi data BZip2.

### *Analisis Data*

Penelitian dilakukan dengan melalui tahapan pertama yaitu studi literatur mengenai susunan algoritma untuk proses kompresi BZip2 (*BWT-MTF- Huffman coding*). Pembuatan beberapa contoh-contoh kedua proses kompresi dan dekompresi, jika perlu dengan simulasi dalam bentuk program dengan menggunakan Maple.

## HASIL PENELITIAN

### *Algoritma Transformasi Burrows-Wheeler (BWT)*

Proses *encoding* menghasilkan sebuah barisan $L$ dan sebuah indeks $s$. Algoritma *decoding* mereproduksi barisan sumber asli dengan memakai $L$ dan sebuah indek $k$.

Input : sebuah untaian simbol-simbol atau huruf $S = s_1, ..., s_n$ di dalam alfabet $\{a_1, a_2, ..., a_n\}$

Output : sebuah untaian simbol-simbol $L$ dan indeks $k$, posisi simbol awal $s_1$ dari $S$ di dalam $L$.

Dalam mengurutkan $L$, dibutuhkan pengerjaan dengan langkah-langkah algoritma dari proses *BWT*. Pada untaian input, tambahkan simbol ~ menjadi simbol awal sehingga $S$=”~”+ $S$. Pertama geser untaian $S$ satu simbol ke kiri secara melingkar. Dengan cara ini, simbol paling kiri dalam larik digeser keluar larik dan kemudian menambahkan kembali dari kanan sehingga simbol menjadi unsur paling kanan. Ulangi penggeseran sebanyak $n - 1$ kali. Bersama untaian awal $S$, hasil pergeseran ini secara berurutan membentuk baris-baris matriks M1 berukuran $n \times n$ dimana $n$ adalah banyaknya simbol dalam larik dan setiap baris dan kolom merupakan sebuah permutasi partikuler $S$. Membentuk matriks M2 dengan mengurutkan baris-baris matriks M1 secara leksikografik Namakan kolom terakhir matriks M2 dengan $L$. $L$ dapat disimpan dalam sebuah larik yang berisi untaian permutasi $S$. Namakan kolom pertama matriks M2 dengan $F$. Output proses *BWT* adalah $BWT(S) = (L)$. Implementasi algoritma *BWT* menampilkan visualisasi seperti (gambar 1).

***Algoritma Move-To-Front* (*MTF*)**

Proses *encoding* (Penurunan *J*)

Input : sebuah untaian $L=BWT(S)$ yang merupakan output proses *BWT* , alfabet *A*dan *k* yaitu indeks simbol awal $s_1$ dalam *L*.

Output : untaian simbol *J* dalam bilangan bulat nonnegatif.

Ilustrasi proses ini diberikan dalam dengan langkah-langkah algoritma dari proses *MTF* berikut: Awalnya, mendaftar alfabet $A= \{A[0], A[1], A[2], ..., A[m]\}$ pada untaian *L* yang selanjutnya disimpan dalam larik yang dimulai dengan indeks 0. Membaca simbol $L[i]$, $i$ =1 dalam *A*, yaitu simbol pada barisan inputan *L*.Jika diperoleh $L[i] = A[j_i]$, $j_i$ =0, 1, 2, .., *m*. Kemudian mengodekan $L[i]$ denga n $j_i$ yaitu dengan menyimpan $j_i$ dalam *Simpan* yaitu *Simpan*[*i*]. Memindahkan $A[j_i]$ dalam alfabet *A* ke depan menjadi larik pertama $A'[0]=A[j_i]$ dan $A'[x]=A[x–1]$, $x$=1, 2, ..., $j_i$ , sehingga diperoleh urutan alfabet dalam larik yaitu *A*'. Membaca simbol selanjutnya *L*[i] untuk $i$ =2. Ulangi langkah 3 s.d 5 sampai semua simbol simbol $L$[i], $i$ =1, 2, .., $n$ telah dikodekan satu demi satu. Output proses *MTF* adalah $MTF(L, k)= (Simpan,A')$. Implementasi algoritma *MFT* menampilkan visualisasi seperti (gambar 2). Dengan cara ini, simbol yang frekuensi kemunculannya lebih dikodekan dengan 0.

***Algoritma Huffman coding***

Adapun masalah kompresi data dirumuskan sebagai berikut:

Input : sebuah untaian $Simpan=MTF(L, k)$ yang merupakan output proses *MTF* dan alfabet dalam larik yaitu *A*'.

Output : untaian-untaian biner, satu untaian untuk setiap simbol dari alfabet sedemikian hingga untaian-untaian biner 0-1 yang diperoleh bisa dibuat sependek mungkin.

Berikut algoritma penyelesaiannya yang lebih rinci: Pertama, menghitung frekuensi dari setiap simbol dari untaian *Simpan*. Menyimpan karakter beserta frekuensi dalam himpunan *KarFr*.

*KarFr*={*Simpan*[1]=frekuensi(*Simpan*[1]),*Simpan*[2]=frekuensi(*Simpan*[2]),...,*Simpan*[*n*]=frekuensi(*Simpan*[*n*])}

Bentuk entri-entri himpunan himpunan tidak sesuai dengan bentuk yang diinginkan: frekuensi dulu baru karakter. Membalik ruas kanan (rhs) setiap entri x ∈ *KarFr* dengan ruas kiri (lhs) entri x tersebut. Misalnya entri "" = 7 menjadi 7 = "". Menyimpan balikan dalam himpunan *FrKar* sehingga secara otomatis anggota akan terurut secara ascending berdasarkan frekuensinya.

*FrKar*={frekuensi(*Simpan*[1])=*Simpan*[1],frekuensi(*Simpan*[2])=*Simpan*[2],...,frekuensi(*Simpan*[*n*])=*Simpan*[*n*]}

Sekarang, pohon Huffman bisa dikontruksi secara rekursif. Jika banyaknya anggota himpunan *FrKar* kurang dari atau sama dengan satu maka pilih dua simbol yang frekuensinya paling kecil x: *Simpan*[*i*] ,y: *Simpan*[*j*]. Sehingga terbentuk subpohon *z*(1) dengan cabang kiri daun x dan cabang kanan daun y dengan frekuensi f[*z*(1)]=f[x]+f[y]. Selanjutnya terbentuk himpunan baru *FKT*, yaitu anggota x dan y diganti dengan *z*(1) dari himpunan *FrKar* sehingga diperoleh:

*FKT*={frekuensi(*Simpan*[1])=*Simpan*[1],frekuensi(*Simpan*[2])= *Simpan*[2],...,
f[*z*(1)] = *z*(1),...,frekuensi(*Simpan*[*n*])=*Simpan*[*n*]}
*FrKar* =*FKT*

Jika banyaknya anggota himpunan *FrKar* sama dengan satu maka anggotanya merupakan akar pohon biner yang terbentuk. Ulangi sampai semua simbol dalam *Simpan* telah menjadi daun. Sehingga terbentuk pohon *Huffman* yang dilanjutkan dengan penelusuran pohon *Huffman* dijelaskan dalam algoritma berikut: Inisialisasi sebuah pohon biner (*Huffman*) *FKT* untuk alfabet ($s_1$, $s_2$, ... , $s_n$) dan string kosong *p*=" ". Penelusuran dimulai dari akar pohon *Huffman* berpindah ke subpohon dibawahnya sampai menjangkau daun. Jika menelusuri subpohon kiri maka diperoleh *p*=p+"0" atau jika menelusuri subpohon kanan maka diperoleh *p*=*p*+"1". Jika telah menjangkau (daun) simbol *s* maka diperoleh kode biner *s*=*p*. Ulangi sampai semua daun telah dijangkau atau semua simbol telah dikodekan dengan kode biner. Tahap terakhir penentuan kode biner pada input string *Simpan* yang dilakukan dengan memetakan setiap simbol dalam string ke kode biner berturut-turut sehingga diperoleh untaian bit 0-1 yang disimpan dalam *KodeBiner*. Implementasi algoritma *Huffman-encoding* menampilkan visualisasi seperti (gambar 3).

## PEMBAHASAN

Dalam pelaksanaan penelitian ini, yakni pada pembuatan implementasinya dalam bentuk program Maple terdapat beberapa Kendala. Salah satu kendala adalah menggunakan salah satu *high-level programming language*, Maple, maka program tidak bisa efisien sebab terlalu banyak memori dari Maple yang digunakan untuk menyajikan tampilan, paket *built-in* dan menu-menu pendukung *user-friendly*.

Faktor lain yang membuat program menjadi agak lambat adalah adanya konstruksi dua matriks di dalam algoritma *BW*. Konstruksi kedua matriks ini sebenarnya bisa dihilangkan dan diganti dengan menggunakan konsep *suffix array*. Di dalam Maple, output program transformasi MTF (yang akan menjadi input dari pengkodean Huffman) memiliki *type*/bentuk daftar (*list*) bilangan-bilangan sehingga harus diubah ke *type* string sebab program yang dibuat untuk pengkodean Huffman hanya menerima input bentuk *string*. Walaupun kendala ini bisa diatasi dengan mengubah bilangan-bilangan 0 sampai 91 menjadi 92 karakter-karakter ASCII yang

*printable* dan berurutan, kendala baru muncul, yaitu tidak ada karakter yang bisa mengganti bilangan 92, 93, …, dan seterusnya.

## KESIMPULAN DAN SARAN

Berdasarkan hasil dan pembahasan dapat disimpulkan bahwa algoritma *BW* cenderung mengumpulkan simbol yang sering muncul dalam teks asli menjadi satu subbarisan di dalam output algoritma. Untuk algoritma *MTF*, menghasilkan daftar (*list*) bilangan-bilangan non-negatif sedemikian sehingga simbol yang kemunculannya lebih sering akan dinyatakan oleh bilangan yang lebih kecil. Sehingga Kombinasi ketiga algoritma *BW*, *MTF* dan *Huffman coding* menghasilkan hasil kompresi teks yang efisien dalam bentuk untaian biner, walaupun lambat. Untuk pengembangan selanjutnya disarankan agar program simulasi yang dibuat menggunakan konsep kombinasi transformasi *BWT*, *DC* dan *Fibonacci Coding* yang merupakan teknik kompresi yang diharapkan bisa memperkecil rasio.

## UCAPAN TERIMA KASIH

Penulis menyampaikan terimakasih kepada Komisi Penasehat Dr. Loeky Haryanto, MS, M.Sc, MA dan Dr.Eng. Armin Lawi, S.Si., M.Eng yang telah memberikan pengarahan dan petunjuk dalam menyelesaikan jurnal ilmiah ini, serta kepada semua pihak yang telah memberikan bantuan dan fasilitas dalam penulisan jurnal ilmiah ini.

## DAFTAR PUSTAKA


Bastys R. (2010). Fibonacci Coding Within The Burrows-Wheeler Compression Scheme. *Electronics and Electrical Engineering-Kaunas:Technologija*, No. 1 (97), pp. 28-32.

Burrows M. & Wheeler DJ. (1994). A Block-Sorting Lossless Data Compression Algorithm. Research report 124. Digital Systems Research Center.

Deorowicz S. (2003). Universal Lossless Data Compression Algorithms, Doctor of Philosophy Dissertation, Silesian University of Technology.

Khalid S. (2006). Introduction to Data Compression 3rd Edition. San Fransisco: Elsevier Inc.

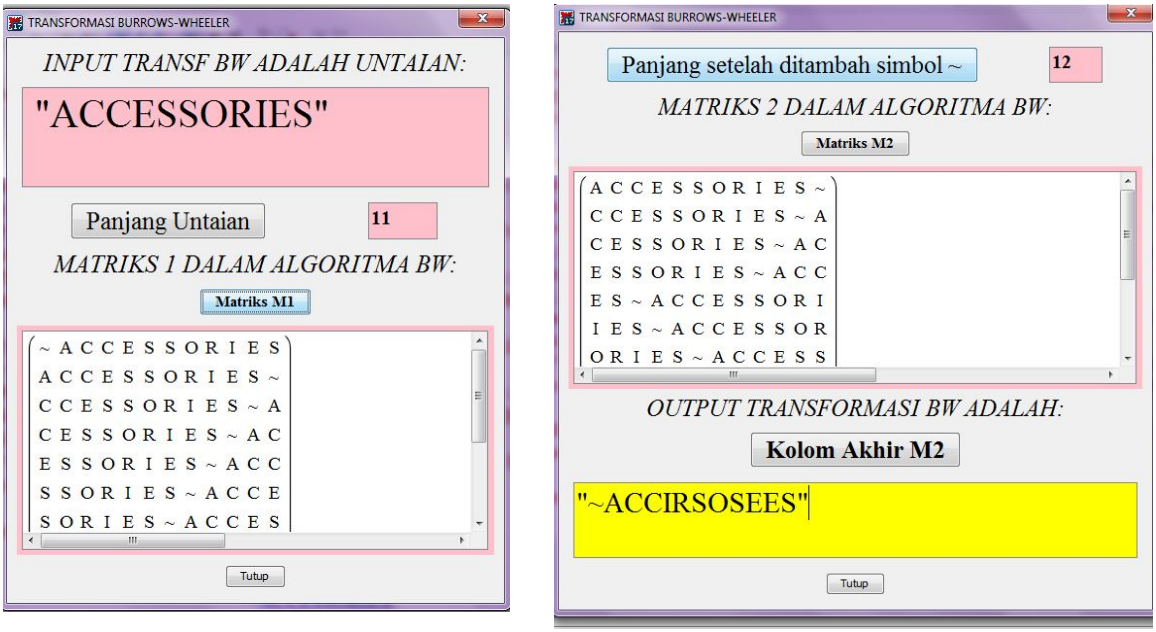
Kumar S. (2012). Elegant Exact String Match Using BWT – FM Index. diakses 01 Maret 2014. http://blog.avadis-ngs.com/2012/04/elegant-exact-string-match-using-bwt-2

Munir R. (2010). Matematika Diskrit (Revisi Keempat). Informatika Bandung.

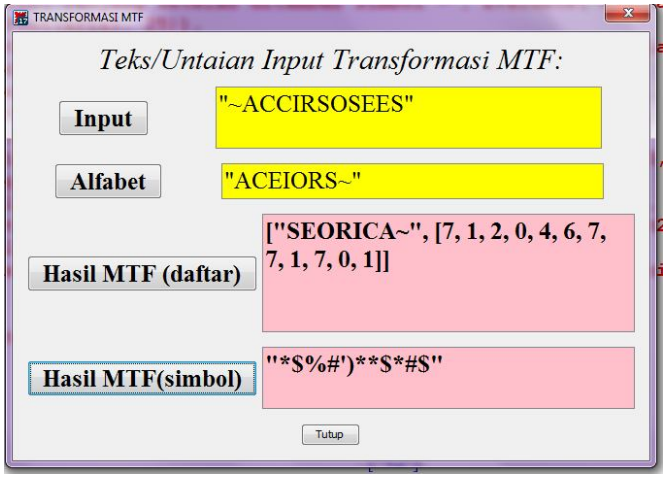
Pu IM. (2006). Fundamental Data Compression. London: Butterworth-Heinemann.

Solomon D. (2007). Data Compression The Complete Reference 4th Edition. London : Springer-Verlag.

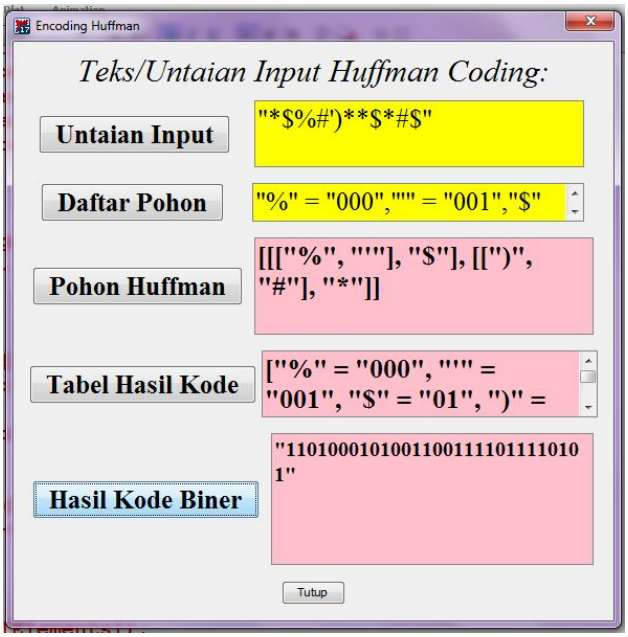
Suarjaya IMAD. (2012). A New Algorithm for Data Compression Optimization. *International Journal of Advanced Computer Science and Applications* (*IJACSA*), Vol. 3, No.8, pp. 14-17.




**Gambar 1. Visualisasi transformasi *BWT***




**Gambar 2. Visualisasi Trasnsformasi *MTF***




**Gambar 3. Visualisasi Trasnsformasi *Huffman-encoding***