Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – KI141502

STEGANOGRAFI AUDIO MENGGUNAKAN METODE LEAST SIGNIFICANT BIT (LSB) DAN KOMPRESI ADAPTIVE HUFFMAN BWT

RAHMAT IRFAN
NRP 5112100008

Dosen Pembimbing I
Henning Titi Ciptaningtyas, S.Kom, M.Kom.

Dosen Pembimbing II
Hudan Studiawan, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

TUGAS AKHIR – KI141502

STEGANOGRAFI AUDIO MENGGUNAKAN METODE LEAST SIGNIFICANT BIT (LSB) DAN KOMPRESI ADAPTIVE HUFFMAN BWT

RAHMAT IRFAN
NRP 5112100008

Dosen Pembimbing I
Henning Titi Ciptaningtyas, S.Kom, M.Kom.

Dosen Pembimbing II
Hudan Studiawan, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESES – KI141502

AUDIO STEGANOGRAPHY USING LEAST SIGNIFICANT BIT (LSB) METHOD AND ADAPTIVE HUFFMAN BWT COMPRESSION

RAHMAT IRFAN
NRP 5112100008

Supervisor I
Henning Titi Ciptaningtyas, S.Kom, M.Kom.

Supervisor II
Hudan Studiawan, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2016

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

STEGANOGRAFI AUDIO MENGGUNAKAN METODE LEAST SIGNIFICANT BIT (LSB) DAN KOMPRESI ADAPTIVE HUFFMAN BWT

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Komputasi Berbasis Jaringan
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh
RAHMAT IRFAN
NRP : 5112 100 008

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Henning Titi Ciptaningtyas, S.Kom, M.Kom. .....
   NIP: 19840708 201012 2 004 (Pembimbing 1)

2. Hudan Studiawan, S.Kom., M.Kom. .....
   NIP: 19870511 201212 1 003 (Pembimbing 2)

SURABAYA
JUNI, 2016

[Halaman ini sengaja dikosongkan]

STEGANOGRAFI AUDIO MENGGUNAKAN METODE LEAST SIGNIFICANT BIT (LSB) DAN KOMPRESI ADAPTIVE HUFFMAN BWT

Nama Mahasiswa	**: Rahmat Irfan**
NRP	**: 5112100008**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Henning Titi Ciptaningtyas, S.Kom, M.Kom.**
Dosen Pembimbing 2	**: Hudan Studiawan, S.Kom., M.Kom.**

Abstrak

Semakin banyak data yang disisipkan pada saat proses steganografi dilakukan, semakin berkurang nilai peak signal to noise (PSNR) dan signal to noise (SNR) dari berkas audio. Hal ini akan membuat proses menyembunyikan data berukuran besar menjadi lebih sulit karena nilai signal to noise yang tergolong kecil. Oleh karena itu untuk mendapatkan nilai PSNR dan SNR yang lebih baik, data dapat diproses terlebih dahulu sebelum disisipkan sehingga berukuran lebih kecil dibandingkan ukuran data awal.

Pada Tugas Akhir ini, dikembangkan program yang dapat menyisipkan dan mengambil informasi pada berkas audio. Aplikasi ini menggunakan teknik steganografi LSB dan teknik kompresi Adaptive Huffman dengan preprocessor BWT sebagai fungsi utamanya.

Hasil yang dari penggabungan kedua teknik tadi adalah berkas audio yang memiliki nilai PSNR dan SNR yang lebih baik dibandingkan dengan hanya menggunakan steganografi saja. Selain itu, perubahan pada kualitas suara berkas hasil tidak terdeteksi ketika didengar oleh telinga manusia.

Kata kunci: BWT, Adaptive Huffman, Steganografi LSB, Waveform.

[Halaman ini sengaja dikosongkan]

AUDIO STEGANOGRAPHY USING LEAST SIGNIFICANT BIT (LSB) METHOD AND ADAPTIVE HUFFMAN BWT COMPRESSION

Nama Mahasiswa	**: Rahmat Irfan**
NRP	**: 5112100008**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Henning Titi Ciptaningtyas, S.Kom, M.Kom.**
Dosen Pembimbing 2	**: Hudan Studiawan, S.Kom., M.Kom.**

Abstract

Embedding too much data in a steganography process will decrease the value of peak signal to noise ratio (PSNR) and signal to noise ratio(SNR) of the audio file. Because of this limitation, hiding an enormous data becomes a very difficult task caused by low signal to noise ratio. In order to increase the peak signal to noise ratio and the signal to noise ratio, all of the data can be processed first before getting embedded to get data with smaller size compared to the original data.

For this project, a program that can embed and extract data from audio file is developed. This application uses LSB steganography method combined with Adaptive Huffman compression method and BWT preprocessing as its main function.

The result from the joining of these methods above is an audio file with better PSNR and SNR value compared to using steganography only by itself. Also, the changes in the quality itself are not detected by human ear.

Keywords: BWT, Adaptive Huffman, LSB Steganography, Waveform.

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji dan syukur ke hadirat Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **"STEGANOGRAFI AUDIO MENGGUNAKAN METODE LEAST SIGNIFICANT BIT (LSB) DAN KOMPRESI ADAPTIVE HUFFMAN BWT".**

Tugas akhir ini dilakukan dalam rangka untuk memenuhi salah satu syarat memperoleh gelar Sarjana Komputer di Program Studi S-1 Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember. Selain itu, tugas akhir ini juga merupakan implementasi dari materi yang telah penulis pelajari selama 4 tahun menjadi mahasiswa.

Penulis juga tak lupa pada kesempatan kali ini mengucapkan rasa terima kasih kepada pihak-pihak yang telah membantu dan memberikan semangat kepada penulis selama penyusunan Tugas Akhir ini, yaitu :

1. Allah SWT dan Nabi Muhammad SAW.
2. Keluarga penulis yang telah memberikan dukungan moral dan material serta do'a yang tak terhingga untuk penulis. Serta selalu memberikan semangat dan motivasi pada penulis dalam mengerjakan Tugas Akhir ini.
3. Ibu Henning Titi Ciptaningtyas, S.Kom, M.Kom. selaku pembimbing I yang telah membantu, membimbing, dan memotivasi penulis dalam menyelesaikan Tugas Akhir ini dengan sabar.
4. Bapak Hudan Studiawan, S.Kom., M.Kom**.** selaku pembimbing II yang juga telah membantu, membimbing, dan memotivasi kepada penulis dalam mengerjakan Tugas Akhir ini.
5. Bapak Dr. Darlis Herumurti, S.Kom., M.Kom. selaku Kepala Jurusan Teknik Informatika ITS.

6. Bapak Radityo Anggoro, S.Kom., M.Sc. selaku koordinator Tugas Akhir, dan segenap dosen Teknik Informatika yang telah memberikan ilmunya.
7. Teman-teman angkatan 2012 yang yang telah membantu, berbagi ilmu, menjaga kebersamaan, dan memberi motivasi kepada penulis.
8. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa pada Tugas Akhir ini masih terdapat banyak kekurangan dan kesalahan. Sehingga, penulis mengharapkan kritik dan saran yang membangun untuk dijadikan bahan perbaikan selanjutnya.

Surabaya, Juni 2016

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR PSEUDOCODE

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1 Latar Belakang

Steganografi berasal dari bahasa Yunani yang berarti ‘tulisan rahasia’. Berbeda dengan kriptografi, steganografi bertujuan untuk menyembunyikan keberadaan sebuah pesan dimana kriptografi akan mengacaknya. Lebih tepatnya, steganografi bertujuan untuk menyembunyikan pesan dalam sebuah data biasa menggunakan cara tertentu sehingga pihak lain tidak mengetahui keberadaan pesan tersebut. Berkas yang akan disisipi selanjutnya akan dikenal dengan berkas *cover* [1].

Steganografi audio LSB (*Least Significant Bit*) merupakan salah satu cara menyembunyikan data ke dalam sebuah berkas *cover* dimana berkas induk merupakan sebuah berkas audio [1]. Namun, semakin banyak LSB yang digunakan untuk menyisipkan data, semakin berkurang kualitas dan nilai *signal to noise* dari sebuah berkas [2].

Untuk mendapatkan nilai *signal to noise* dan *peak signal to noise* yang lebih baik. Data dapat diproses terlebih dahulu sehingga berukuran lebih kecil dibandingkan berkas aslinya. Terdapat beberapa cara yang dapat digunakan untuk mengurangi ukuran data yang akan disisipkan, salah satunya yaitu dengan menggunakan sebuah algoritma kompresi. Algoritma kompresi ini dapat mengurangi data redundan dan menurunkan ukuran data yang disimpan. Selain itu, kompresi data akan mengubah informasi menjadi kode yang redundan sehingga dapat menjaga kerahasiaan data dengan lebih baik. Penggabungan steganografi dengan kompresi data tersebut dapat menghasilkan nilai *signal to noise* dan *peak signal to noise* yang lebih baik [1].

Teknik kompresi Adaptive Huffman merupakan teknik yang dikembangkan dari teknik kompresi Huffman dimana karakter

pada input diubah menjadi *codeword* dengan panjang sesuai dengan statistik kemunculan huruf tersebut. Namun Adaptive Huffman tidak melihat input secara keseluruhan untuk membuat pohon *codeword*, melainkan membuat dan memperbaharui pohon saat proses *encoding* dan *decoding* berlangsung [3].

Preprocessing BWT (Burrow-Wheeler Transform) merupakan teknik *preprocessing* dimana karakter dari teks input akan di permutasi sehingga menghasilkan huruf-huruf yang sama cenderung akan berkelompok pada posisi tertentu pada teks output [4]. Penggunaan BWT diharapkan dapat membantu teknik kompresi mengurangi ukuran berkas yang akan disisipkan.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana membuat aplikasi yang dapat menyimpan pesan dalam berkas audio.
2. Bagaimana membuat aplikasi yang dapat mengambil pesan yang telah disimpan dalam berkas audio.
3. Bagaimana membuat aplikasi yang dapat melakukan kompresi terhadap pesan sebelum disimpan dalam berkas audio.
4. Bagaimana performa kompresi aplikasi tersebut jika dibandingkan dengan performa kompresi lain

1.3 Batasan Masalah

Batasan dalam Tugas Akhir ini, yaitu:

1. Aplikasi ini berbasis desktop dengan bahasa pemrograman C++.
2. Huruf yang didukung hanyalah huruf ASCII (*American Standard Code for Information Interchange*).
3. Format audio yang didukung adalah WAV (*Waveform Audio File*).
4. Algoritma steganografi yang digunakan adalah LSB

5. Algoritma kompresi data yang digunakan adalah Adaptive Huffman dengan *preprocessing* BWT.

1.4 Tujuan

Tujuan tugas akhir ini adalah menganalisa dan membandingkan kualitas antara audio yang disisipi pesan menggunakan teknik steganografi tanpa kompresi dengan audio yang disisipi pesan menggunakan teknik steganografi dengan kompresi dan *preprocessing*.

1.5 Manfaat

Dengan dibuatnya Tugas Akhir ini diharapkan bisa menemukan nilai terbaik antara kualitas berkas audio dan banyaknya bit yang digunakan untuk menyembunyikan berkas pesan pada berkas audio tersebut.

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Proposal Tugas Akhir yang diajukan memiliki gagasan yang sama dengan Tugas Akhir ini. Penyusunan proposal Tugas Akhir dilaksanakan untuk merumuskan masalah serta melakukan penetapan rancangan dasar dari sistem yang akan dikembangkan dalam pelaksanaan Tugas Akhir ini.

2. Studi literatur
   Pada tahap ini dilakukan pemahaman informasi dan literatur yang diperlukan untuk tahap implementasi program. Tahap ini diperlukan untuk membantu memahami penggunaan komponen-komponen terkait dengan sistem yang

akan dibangun, antara lain: steganografi secara umum, steganografi audio LSB, algoritma BWT, algoritma Adaptive Huffman, PSNR (*Peak Signal To Noise*), SNR(*Signal To Noise*), serta *Waveform Audio File*.

3. Analisis dan perancangan perangkat lunak

   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini mendefinisikan alur dari implementasi. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini. Pada tahapan ini dibuat rancangan dasar dari sistem. Serta dilakukan desain fungsi yang akan dibuat yang ditunjukkan melalui diagram alir. Fungsi utama yang akan dibuat pada tugas akhir ini meliputi fungsi *preprocessing* BWT, reversi BWT, kompresi Adaptive Huffman, dekompresi Adaptive Huffman, penyisipan LSB, dan ekstraksi LSB.

4. Implementasi perangkat lunak

   Implementasi perangkat lunak merupakan tahap membangun rancangan program yang telah dibuat. Pada tahap ini akan direalisasikan mengenai rancangan apa saja yang telah didefiniskan pada tahap sebelumnya. Fungsi yang ada pada tahap ini merupakan fungsi hasil implementasi dari tahap analisis dan perancangan perangkat lunak.

5. Pengujian dan evaluasi

   Pada tahap ini dilakukan uji coba pada data yang telah dikumpulkan. Tahap ini digunakan untuk mengevaluasi kinerja program serta mencari masalah yang mungkin timbul saat program dievaluasi serta melakukan perbaikan jika terdapat kesalahan pada program.

6. Penyusunan buku Tugas Akhir.
   Pada tahap ini disusun buku yang memuat dokumentasi mengenai perancangan, pembuatan serta hasil dari implementasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu perumusan masalah, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

Bab II Tinjauan Pustaka

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini. Dasar Teori yang digunakan adalah steganografi secara umum, steganografi audio LSB, algoritma BWT, algoritma Adaptive Huffman, PSNR, SNR, serta *Waveform Audio File*.

Bab III Analisis dan Perancangan Perangkat Lunak

Bab ini berisi tentang rancangan sistem yang akan dibangun dan disajikan dalam bentuk diagram alir. Fungsi utama yang akan dibuat pada tugas akhir ini meliputi fungsi preprocessing BWT, reversi BWT, kompresi Adaptive Huffman, dekompresi Adaptive Huffman, penyisipan LSB, dan ekstraksi LSB.

Bab IV Implementasi

Bab ini membahas mengenai implementasi dari rancangan yang telah dibuat pada bab sebelumnya. Penjelasan berupa pseudo*code* yang digunakan untuk proses implementasi.

Bab V Uji Coba Dan Evaluasi

Bab ini menjelaskan mengenai kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari perangkat lunak yang telah dibuat sesuai dengan data yang diujikan.

Bab VI Kesimpulan Dan Saran

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang telah dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

BAB II
TINJAUAN PUSTAKA

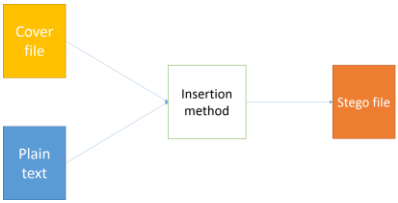
Bab ini berisi penjelasan teori-teori yang berkaitan dengan algoritma yang diajukan pada pengimplementasian program. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 Steganografi

Steganografi adalah proses menyembunyikan informasi dalam informasi lainnya. Tujuan proses ini adalah agar keberadaan informasi rahasia tersebut tidak dapat diketahui oleh orang lain. Media digital dapat digunakan dalam steganografi diantaranya adalah suara, gambar, dan video [1].

Terdapat dua berkas yang diperlukan ketika akan melakukan steganografi, yaitu berkas *cover* dan berkas *message*. Sementara hasil proses ini berupa berkas *cover* yang telah disisipi oleh berkas pesan dan disebut berkas *stego* [2].

Proses steganografi terdiri dari dua macam proses yaitu *insertion* dan *extraction*. Pada Gambar 2.1, terlihat bahwa tahap *insertion* menerima berkas *cover* dan berkas *message* sebagai input dan menghasilkan berkas *stego* sebagai output. Sementara pada Gambar 2.2 dapat dilihat bahwa tahap *extraction* menerima berkas *stego*, dan menghasilkan berkas *message* sebagai output.



Gambar 2.1 Proses Insertion

Pada tahap *insertion*, berkas *message* akan dimasukkan ke dalam berkas *cover* menggunakan metode tertentu sedemikian rupa sehingga nantinya berkas *stego* akan tetap dapat dibuka seperti layaknya membuka berkas *cover*.

Gambar 2.2 Proses Extraction

Tahap *extraction* akan memproses isi berkas *stego* dan mengeluarkan informasi yang telah dimasukkan kedalam berkas *stego* menjadi sebuah berkas baru.

Dalam memproses isi file *stego*, terdapat *header* yang berisi informasi mengenai berkas yang telah dimasukkan. *Header* ini berguna dalam menentukan parameter-parameter yang digunakan ketika proses *extraction* berlangsung.

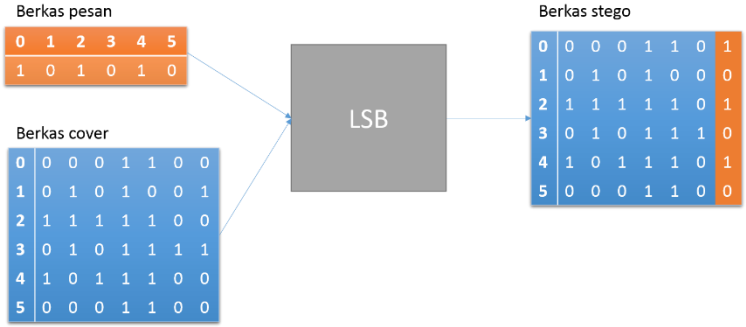
Steganografi digunakan sebagai media pengiriman pesan rahasia kepada orang lain tanpa diketahui oleh pihak penyadap keberadaannya dengan cara menyembunyikannya ke dalam berkas yang tidak diduga oleh orang lain, sehingga aspek kerahasiaan dapat dijaga.

Proses steganografi merupakan proses utama dalam implementasi program. Proses ini akan dijalankan setelah input telah diproses oleh algoritma kompresi dan akan mengeluarkan berkas stego sebagai output.

2.2 Steganografi LSB

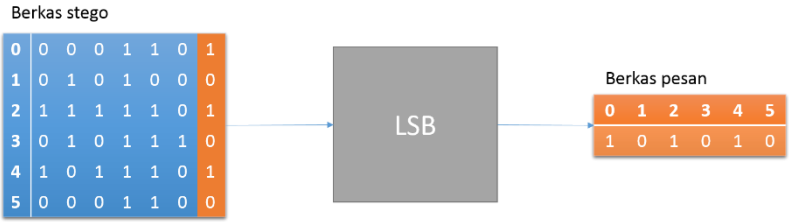
Steganonografi LSB merupakan metode paling sederhana yang digunakan untuk menyembunyikan data. Dimana *bit* dengan posisi terkecil pada sampel dari berkas *cover* akan diubah sesuai dengan *bit* pesan yang akan dimasukkan [5]. Penggunaan posisi *bit*

terkecil bertujuan agar perubahan yang dihasilkan tidak terdengar oleh pendengar [6]. Selain itu, sistem pendengaran manusia memiliki kelemahan sehingga perubahan yang sangat kecil dapat tertutupi oleh suara keras dan pendeteksian terhadap perubahan tersebut dapat dihindari.



Gambar 2.3 Proses Insertion LSB

Proses *insertion* steganografi LSB, seperti yang terlihat pada Gambar 2.3, dilakukan dengan cara menggantikan secara langsung *bit* posisi terakhir pada sampel dengan *bit* yang terdapat didalam pesan. Jumlah kolom *bit* yang dapat diubah untuk musik pada umumnya adalah 3 [2]. Apabila jumlah kolom yang diubah melebihi dari nilai tersebut, maka perubahan akan mengakibatkan munculnya derau yang dapat dideteksi oleh pendengar.



Gambar 2.4 Proses Extraction LSB

Ekstraksi data pada teknik LSB dilakukan dengan membaca *bit* posisi terkecil pada sampel dan melakukan menuliskan *bit* tersebut pada sebuah berkas baru.

Algoritma LSB digunakan sebagai algoritma steganografi sebab algoritma ini memiliki cara kerja yang sederhana dimana *bit* pesan langsung dimasukkan ke dalam sampel berkas *cover* [2]. Sehingga peningkatan dapat dilakukan dengan cara memanipulasi data yang akan dimasukkan sebelum proses LSB dilakukan. Hal ini mengakibatkan proses steganografi akan didahului oleh proses-proses lainnya ketika melakukan tahap *insertion* dan akan menghasilkan data yang perlu diproses lagi ketika melalui tahap *extraction*.

2.3 Preprocessing BWT

Preprocessing adalah proses yang dilakukan pada saat sebelum proses kompresi utama dilakukan. *Preprocessing* dilakukan untuk mendapatkan hasil kompresi yang lebih baik [3].

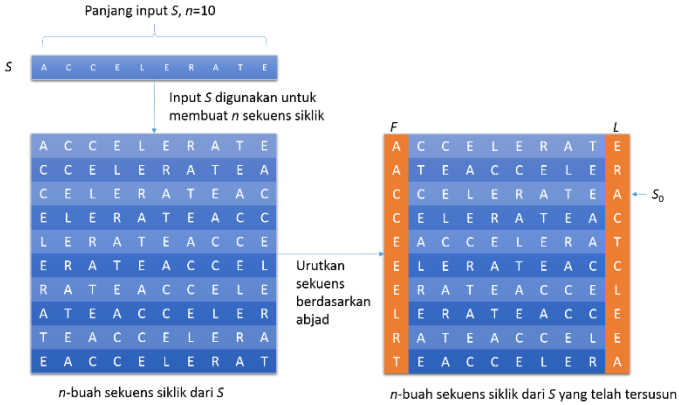
Preprocessing BWT merupakan salah satu teknik *preprocessing* dimana simbol input *S* dengan panjang *n* akan ditransformasikan sehingga menghasilkan output *L* yang akan memiliki karakteristik [4]:

1. Akan terdapat simbol-simbol yang berkonsentrasi pada tempat-tempat tertentu pada *L*. Sehingga simbol-simbol yang berkonsentrasi tersebut lebih mudah di kompresi.
2. Memungkinkan untuk melakukan transformasi reversi yang akan menghasilkan output *S*.

Proses transformasi yang dilakukan pada bwt adalah membuat sekuens simbol sebanyak *n*-1 dari *S*. Dimana setiap sekuens berisi pergeseran siklik karakter dari *S*. Selanjutnya, semua sekuens tersebut di urutkan berdasarkan urutan abjad pada kolom pertama hingga kolom ke-*n*. Sekuens baru *L* dengan panjang *n* lalu akan dibuat dan berisi semua huruf pada kolom ke-*n* pada sekuens-sekuens yang telah diurutkan sebelumnya. Sekuens ini

dan posisi karakter pertama dari sekuens *S* pada sekuens *L* merupakan output dari proses ini.

Contoh proses ini dapat dilihat pada Gambar 2.5. Pada gambar, terlihat kata *Accelerate* dengan *n*=10 dibuat menjadi sekuens siklik dengan jumlah n. Selanjutnya sekuens ini diurutkan sehingga sekuens menjadi berurutan sesuai urutan abjad. *L* kemudian dapat diambil dari kolom terakhir dan S_0 dapat diketahui berdasarkan lokasi tempat karakter awal sekuens pada L.

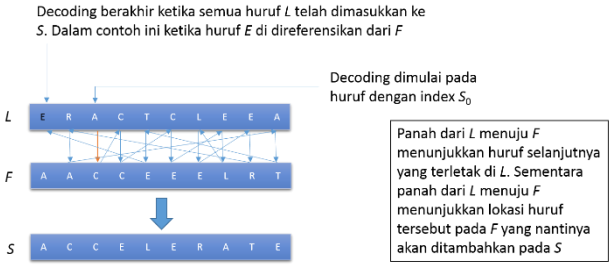


Gambar 2.5 Proses Transformasi BWT

Proses reversi transformasi memerlukan 3 komponen yaitu matriks *L*, matriks *F* yang berisi kolom pertama dari sekuens siklik awal, dan indeks S_0 yang merupakan indeks posisi huruf pertama dari *S* pada matriks *L*. Matriks *F* dapat direkonstruksi dengan cara mengurutkan matriks *L* sesuai urutan abjad. Proses dimulai dengan S_0 yang digunakan untuk menentukan posisi huruf pada matriks *L* dan menentukan huruf selanjutnya pada matriks *F*. Huruf pada matriks *F* kemudian dicari indeksnya pada matriks *L*. Indeks huruf pada matriks *L* lalu akan digunakan untuk menentukan huruf selanjutnya pada matriks *F*. Proses ini diulang hingga rekonstruksi sekuens *S* selesai.

Contoh proses ini dapat dilihat pada Gambar 2.6. Pada contoh ini, *F* didapatkan dengan cara mengurutkan karakter pada

L. Selanjutnya, S_0 digunakan untuk menentukan posisi karakter pertama pada *L*. S_0 juga digunakan untuk menentukan posisi karakter selanjutnya pada *F*. Selanjutnya karakter pada *F* digunakan untuk mencari karakter tersebut pada *L* dengan syarat jumlah kemunculan karakter tersebut sama dengan jumlah kemunculan karakter pada *L*. Sebagai contoh indeks huruf C kedua pada *L* sama dengan indeks huruf E pada *F*. Huruf selanjutnya yang perlu dicari pada *L* adalah huruf E kedua, karena huruf E pada *F* memiliki jumlah kemunculan yang sama dengan huruf E kedua pada *L* yaitu 2. Proses dihentikan ketika semua karakter pada *L* telah dimasukkan ke dalam *S* dengan urutan yang benar.



Gambar 2.6 Proses Reversi BWT

Transformasi BWT digunakan karena dapat mengubah pesan menjadi lebih mudah untuk dikompresi [3]. Oleh karena itu, transformasi ini akan dijalankan sebelum berkas input masuk ke proses kompresi data.

2.4 Kompresi Data

Kompresi data merupakan ilmu di bidang teknik informatika yang mempelajari tentang cara merepresentasikan data dalam bentuk yang lebih padat. Umumnya kompresi digunakan untuk mengurangi ukuran berkas sebelum dilakukan penyimpanan atau pengiriman pada berkas tersebut [4].

Terdapat dua jenis metode kompresi yaitu *lossless* dan *lossy*. *Lossless* merupakan jenis kompresi dimana tidak ada informasi

yang hilang pada saat kompresi dilakukan sehingga bersifat reversibel. Sementara lossy merupakan jenis kompresi dimana terdapat detail tidak penting yang dihilangkan saat proses kompresi dilakukan. Hal ini menyebabkan tidak memungkinkan untuk mendapatkan berkas yang sama persis dengan berkas asli pada saat proses dekompresi dilakukan atau disebut dengan irreversibel [4].

Kualitas hasil kompresi *lossless* dapat diukur dengan membandingkan berkas hasil kompresi dan berkas asli. Salah satu cara yang paling sederhana yaitu dengan menggunakan rasio kompresi [4]. Rumus rasio kompresi dapat dilihat pada rumus di bawah ini.

$$Rasio\ Kompresi = \frac{ukuran\ setelah\ dikompresi}{ukuran\ sebelum\ dikompresi}$$

2.5 Kompresi Adaptive Huffman

Algoritma Adaptive Huffman merupakan pengembangan dari algoritma huffman yang merupakan sebuah teknik kompresi dimana semua simbol seperti karakter ASCII akan diubah menjadi *codeword* dengan panjang yang tegantung dari frekuensi kemunculan simbol. Adaptive Huffman tidak melihat input secara keseluruhan untuk membuat pohon *codeword*, melainkan membuat dan memperbaharui pohon saat proses *encoding* dan *decoding* berlangsung [3].

Output dari algoritma Adaptive Huffman terdiri dari dua jenis *codeword* yaitu huruf biasa dan Huffman *codeword*. Penggunaan huruf biasa sebagai *codeword* hanya digunakan pada saat simbol baru ditemui untuk pertama kali nya. Namun, penggunaan gabungan dari huruf biasa dan Huffman *codeword* akan menjadi masalah pada saat *decoding*. Oleh karena itu simbol khusus digunakan untuk mengganti mode *decoding* pada saat huruf ditemukan [3].

Proses kompresi dimulai dengan pohon kode Huffman yang kosong. Pada saat simbol pertama dimasukkan, maka simbol akan langsung dituliskan sebagai huruf pada output. Selanjutnya simbol akan masuk kedalam pohon dan sebuah kode akan dipasangkan dengan simbol tersebut. Ketika simbol yang sama muncul kembali, maka frekuensi simbol pada pohon akan di naikkan.

Proses dekompresi mirip dengan proses kompresi dimana simbol pertama kali akan dibaca lalu dimasukkan ke dalam pohon. Perbedaan pada dekompresi adalah perlunya menentukan apakah simbol tersebut merupakan huruf atau sebuah Huffman *codeword*. Apabila simbol khusus ditemui, maka kode akan dibaca sebagai huruf. Proses ini akan mengeluarkan berkas utuh yang selanjutnya dapat digunakan pada proses reversi transformasi BWT.

Pemilihan Adaptive Huffman sebagai metode kompresi digunakan untuk membandingkan performa kompresi Adaptive Huffman dengan performa kompresi lainnya dalam memproses berkas pesan yang selanjutnya akan dimasukkan ke dalam berkas *cover*. Oleh karena itu proses ini akan dijalankan sebelum proses steganografi akan dijalankan.

2.6 Waveform Audio File (WAV)

Waveform audio file merupakan format standar untuk audio tidak terkompresi yang dikembangkan oleh Microsoft dan IBM. Waveform menyimpan data suara atau dikenal dengan sampel secara berurutan dengan jumlah *bit* atau resolusi yang dapat ditentukan, baik dengan 1 saluran (mono) maupun lebih dari 1 saluran, seperti yang terlihat pada Gambar 2.7.

Waveform Audio file akan digunakan sebagai cover steganografi lsb, hal ini disebabkan oleh beberapa hal diantara lain:

1. Pendengaran manusia terdapat kelemahan dimana penerimaan suara-suara keras dapat menutupi penerimaan suara-suara kecil [7]. Sehingga derau dengan nilai rendah dapat dimasukkan terutama pada berkas yang memiliki nilai yang beragam [7].

2. Karena karakteristik Waveform yang bersifat tidak dikompresi, maka akan terdapat banyak nilai suara yang akan diabaikan manusia [8]. Hal ini dapat digunakan untuk menutupi keberadaan derau yang dimasukkan ke dalam berkas.

Sampel 1	Sampel 2	Sampel 3	Sampel 4
Saluran 0	Saluran 0	Saluran 0	Saluran 0

Susunan data suara pada berkas waveform dengan 1 kanal

Sampel 1		Sampel 2	
Saluran 0	Saluran 1	Saluran 0	Saluran 1

Susunan data suara pada berkas waveform dengan 2 kanal

Gambar 2.7 Struktur data Waveform

2.7 *Peak Signal to Noise Ratio* (PSNR)

Peak Signal to Noise Ratio (PSNR) adalah nilai yang digunakan untuk membandingkan kemiripan antara berkas hasil rekonstruksi, dalam hal ini berkas *stego*, dengan berkas aslinya [4]. Rumus untuk menghitung nilai PSNR adalah sebagai berikut:

$$PSNR = 20 \cdot \log_{10}(MAX) - 10 \cdot \log_{10}(MSE)$$

Dimana MAX menunjukkan nilai terbesar yang dapat dicapai dan *Mean Square Error* (MSE) adalah nilai rata-rata penyimpangan pangkat dua antara kedua berkas. Semakin mirip kedua berkas (*stego* dan *cover*) yang diproses, maka semakin kecil nilai MSE yang dihasilkan dan akibatnya semakin besar nilai PSNR yang dihasilkan [4]. MSE dihitung menggunakan rumus berikut:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

Dimana m adalah jumlah sampel dan n adalah jumlah saluran. I adalah nilai sampel pada berkas *cover* dan K adalah nilai sampel pada berkas *stego.*

Perhitungan PSNR digunakan karena dapat menunjukkan kualitas dari sebuah berkas *stego*. Proses perhitungan nilai MSE dan PSNR akan dilakukan pada saat penyisipan *bit* dilakukan.

2.8 *Signal To Noise Ratio* (SNR)

Signal To Noise Ratio (SNR) adalah nilai yang menunjukkan intensitas dari derau pada berkas *cover* yang telah disisipi oleh data rahasia [9]. Untuk menjaga kualitas dari suara, nilai SNR harus berada di atas 30 dB [9]. Rumus perhitangan SNR adalah sebagai berikut:

$$SNR = \frac{\sum_{i=0}^{N-1} y^2}{\sum_{i=0}^{N-1} y^2 - \sum_{i=0}^{N-1} y'^2}$$

Dimana y adalah sampel pada berkas asli dan y' adalah sampel pada berkas *stego.*

Penggunaan SNR bertujuan untuk mengetahui tingkat derau pada sebuah berkas stego. Proses perhitungan nilai SNR akan dilakukan pada saat penyisipan *bit* dilakukan

BAB III
ANALISIS DAN PERANCANGAN PERANGKAT LUNAK

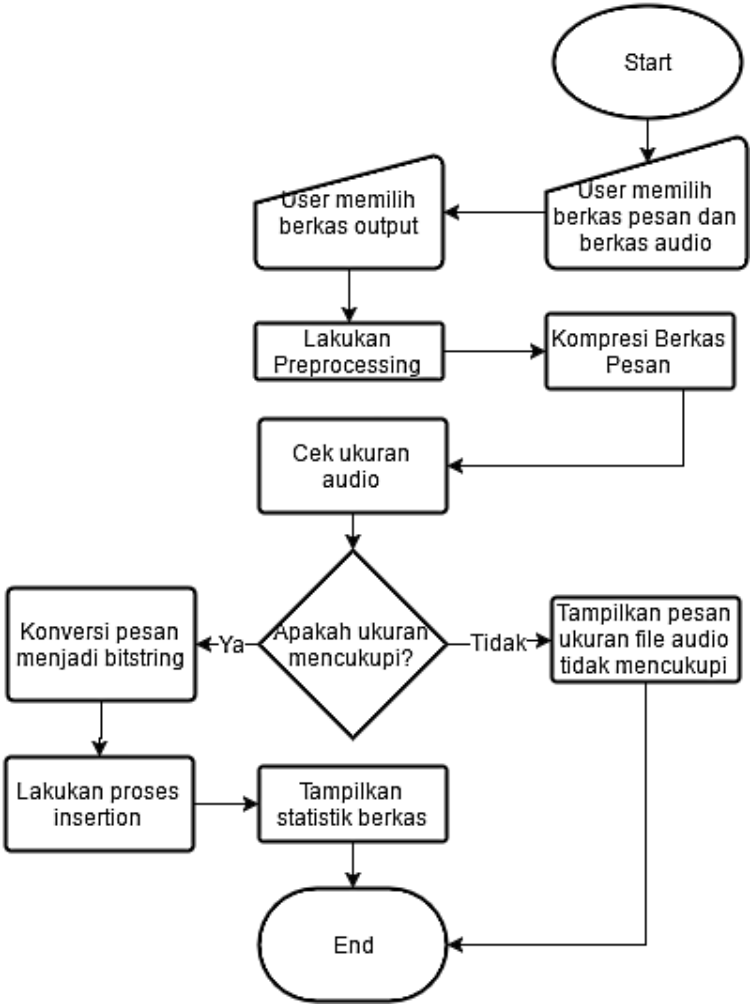
Pada bab ini akan dijelaskan perancangan perangkat lunak yang dibuat. Perancangan akan dibagi menjadi dua tahapan utama, yaitu perancangan tahap *insertion* dan perancangan tahap *extraction*. Pada bab ini juga akan dijelaskan mengenai gambaran umum setiap proses utama program dalam diagram alir beserta penjelasannya.

3.1 Perancangan Tahap *Insertion*

Pada tahap *insertion* terbagi menjadi 3 proses utama, yaitu proses *preprocessing* BWT, proses kompresi Adaptive Huffman, dan proses *insertion* LSB. Semua proses tersebut terhubung melalui 1 fungsi utama yang dijelaskan pada Gambar 3.1.

Proses utama dimulai pada saat program menerima lokasi berkas pesan. Variabel lokasi berkas pesan tersebut digunakan untuk melakukan *preprocessing* BWT. Hasil proses *preprocessing* BWT ini kemudian dikompresi menggunakan algoritma kompresi Adaptive Huffman. Berkas hasil kompresi selanjutnya diperiksa untuk memastikan ukuran berkas tidak melebihi jumlah sampel pada berkas *cover*. Selanjutnya, isi berkas dikonversi menjadi *bitstring* dan digabungkan dengan *bitstring* dari *header* yang telah dibuat. Hasil penggabungan ini kemudian menjadi input dari proses *insertion* LSB dan menghasilkan berkas *stego* sebagai output.

Pada akhir proses, statistik proses akan ditampilkan. Statistik yang akan ditampilkan berupa nilai PSNR, SNR, rasio kompresi, *frame* terpakai, ukuran pesan yang telah dikompresi, serta waktu pengujian yang digunakan. Selain itu, karakteristik berkas input seperti jumlah *frame*, jumlah saluran dan ukuran pesan juga ikut ditampilkan.

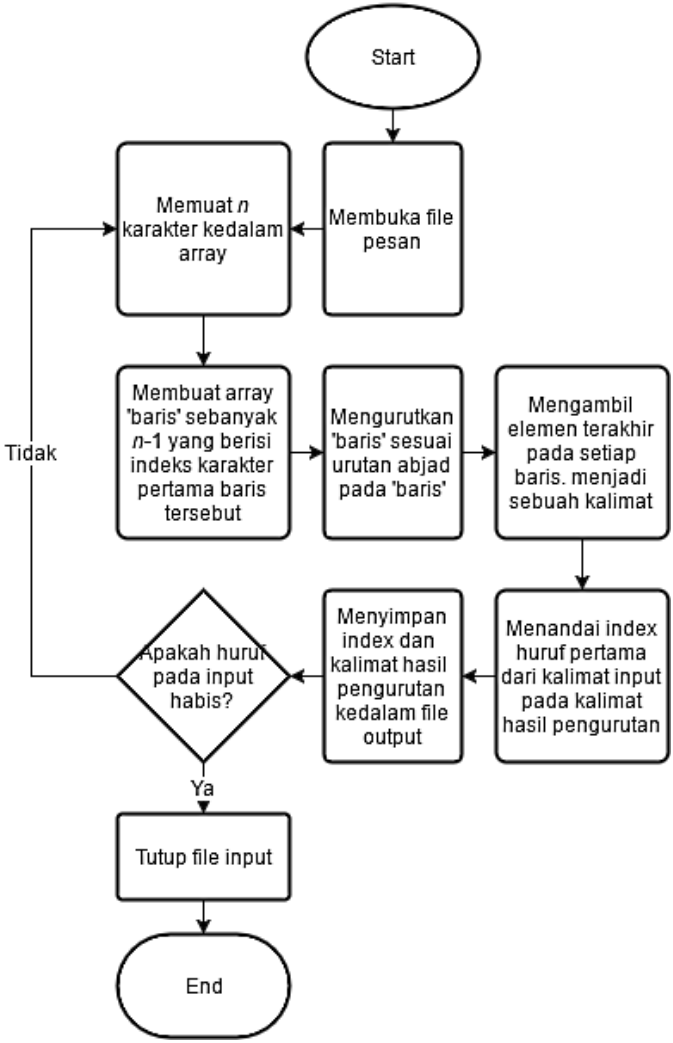


Gambar 3.1 Proses Insertion

3.1.1 Proses *Preprocessing* BWT

Proses *preprocessing* BWT adalah proses transformasi *array* input *S* yang dilakukan sebelum berkas pesan dikompresi

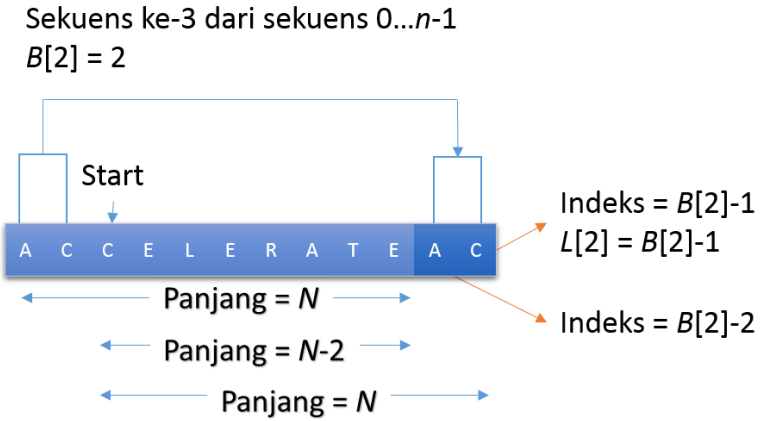
oleh program. Transformasi dilakukan melalui proses sesuai diagram alir pada Gambar 3.2 di bawah ini.



Gambar 3.2 Transformasi BWT

Proses dimulai dengan mengambil karakter pesan sebanyak *n* dan menyimpannya dalam *array S*. Selanjutnya, *array* sekuens *B* dengan ukuran *n* dibuat. *Array* ini berisi indeks karakter ke-0 dari setiap sekuens. Oleh karena itu, *array* akan diinisiasi dengan nilai 0-*n*. Sekuens yang dihasilkan dari *array B* kemudian dibandingkan satu sama lainnya untuk selanjutnya diurutkan sesuai urutan abjad. Setiap huruf ke-*n* pada sekuens kemudian disimpan dalam sebuah *array* baru bernama *L*. Indeks dari huruf ke-0 *array S* pada *array L* kemudian dientukan. Indeks dan semua huruf pada *array L* selanjutnya disimpan di berkas sementara. Proses ini diulang selama masih ada pesan yang tersisa.

Proses transformasi BWT yang sebenarnya terjadi pada saat pengurutan *array* sekuens *B* sebagai tempat untuk menyimpan indeks karakter pertama dari semua sekuens siklik. Sekuens-sekuens tidak disimpan sebagai *array* baru karena hal tersebut akan memakan memori yang besar dimana diperlukan *array* berukuran *n* karakter sebanyak *n* buah.



Gambar 3.3 Contoh sekuens dengan ujung sekuens (karakter ke-n) berada dibelakang karakter awal

Proses pengurutan *array B* ini tidak dapat menggunakan algoritma pengurutan biasa. Hal ini disebabkan oleh keadaan dimana pada setiap sekuens dengan indeks karakter awal (nilai dari $B[i]$, dengan i sebagai nomor sekuens) selain 0, semua karakter dari posisi n+1 hingga n+$B[i]$ (dimana panjang sekuens menjadi n) terletak pada *array S* dengan indeks sebelum $B[i]$. Sebagai contoh, pada Gambar 3.1 terlihat bahwa sekuens ke-3 dengan posisi awal huruf 'C' memiliki 2 huruf terakhir terletak pada posisi sebelum huruf 'C' itu sendiri. Oleh karena itu, diperlukan sebuah fungsi pembanding khusus yang dapat berpindah ke awal *array S* ketika indeks karakter yang dibandingkan telah mencapai ujung *array S* dan berhenti membandingkan ketika panjang karakter yang diurutkan telah mencapai *n*.

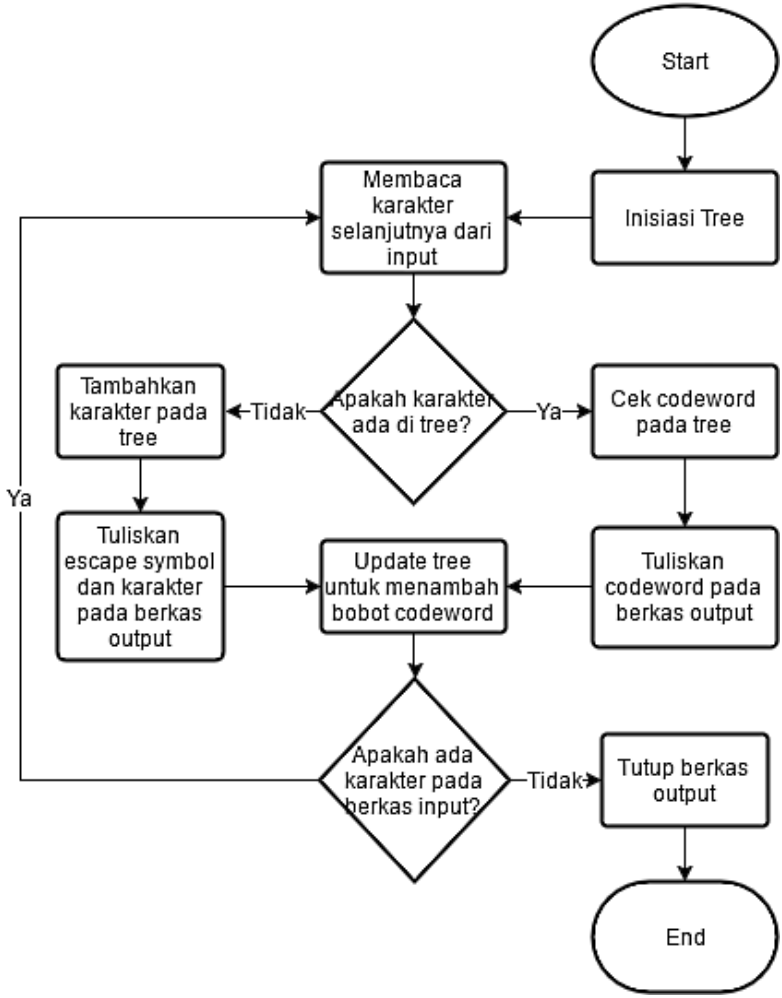
Proses mendapatkan karakter pada *array L* dapat dilakukan dengan rumus:

$$L[i] = \begin{cases} n, & B[i] = 0 \\ i - 1, & B[i] = 1 \dots n - 1 \end{cases}$$

Untuk mendapatkan indeks dari karakter pertama *array S*(indeks karakter S_0) pada *array L*, dapat dilakukan dengan cara mencari indeks salah satu elemen pada *array B* yang berisi nilai 1, dimana karakter awal diputar ke posisi ke-*n*. Nilai indeks S_0 ini beserta isi dari *array L* selanjutnya disimpan ke dalam file sementara. Proses diulangi hingga semua karakter pesan telah diproses.

3.1.2 Proses Kompresi Adaptive Huffman

Proses kompresi Adaptive Huffman adalah proses kompresi dimana pohon Huffman diperbaharui pada saat kompresi berlangsung, sehingga *codeword* pada simbol yang sama tidak selalu sama akibat perubahan struktur pohon. Proses kompresi dilakukan melalui proses yang ditunjukkan pada Gambar 3.4 berikut ini.



Gambar 3.4 Kompresi Adaptive Huffman

Proses dimulai dengan membuka berkas hasil preprocessing. Kemudian inisiasi *tree* dengan *leaf* kosong dilakukan. Selanjutnya karakter akan dibaca dari berkas. Cek apakah karakter tersebut ada pada *tree*. Jika tidak ditemukan,

karakter tersebut akan ditambahkan ke dalam *tree* dengan bobot bernilai nol dan akan dituliskan langsung bersama dengan *escape character* pada berkas output. Jika karakter tersebut ditemukan, *codeword* dari karakter akan dituliskan pada berkas output. Selanjutnya, *tree* akan diperbaharui dengan cara menambahkan bobot pada *codeword* karakter. Proses akan diulangi hingga semua karakter pada berkas pesan diproses.

Proses *update* pada *tree* dilakukan dengan menambahkan bobot pada *leaf* dari *tree*. Pada saat penambahan bobot, pertukaran komponen dapat terjadi untuk memastikan bahwa karakter dengan bobot lebih tinggi memiliki *codeword* lebih pendek dibandingkan karakter yang memiliki bobot rendah. Hasil dari proses ini adalah berkas sementara yang telah di kompresi.

3.1.3 Proses *Insertion* LSB

Proses *insertion* LSB merupakan proses utama dari keseluruhan program ini dimana berkas hasil proses kompresi akan dimasukkan ke dalam berkas *cover* untuk menghasilkan berkas *stego*. Proses *insertion* ini dilakukan dengan sesuai dengan alur diagram pada Gambar 3.5.

Proses dimulai dengan memuat sebanyak *n* buah *frame* ke dalam *buffer*. Selanjutnya bit pada 1 *frame* dengan panjang *k* bit akan digantikan dengan bit message dimulai dengan posisi terkecil hingga posisi *k*. Proses diulang pada *frame* selanjutnya hingga seluruh bit pesan telah dimasukkan. Selama pengerjaan, *buffer* akan dituliskan pada berkas *stego* dan data baru akan dimuat jika semua elemen pada *buffer* telah dimodifikasi. Sisa *frame* pada berkas *cover* kemudian dituliskan pada berkas *stego*.

Proses *insertion* dilakukan dengan cara mengecek terlebih dahulu *bit* pada kolom yang akan diubah menggunakan operasi *bitwise*. Jika *bit* bernilai 1 dan *bitstring* bertanda ‘0’, maka operasi *invert* pada *bit* tersebut akan dilakukan sehingga *bit* tersebut menjadi 0. Proses yang sama juga dilakuka untuk *bitstring* bertanda ‘1’ dan *bit* bernilai 0.

Gambar 3.5 Proses Insertion LSB

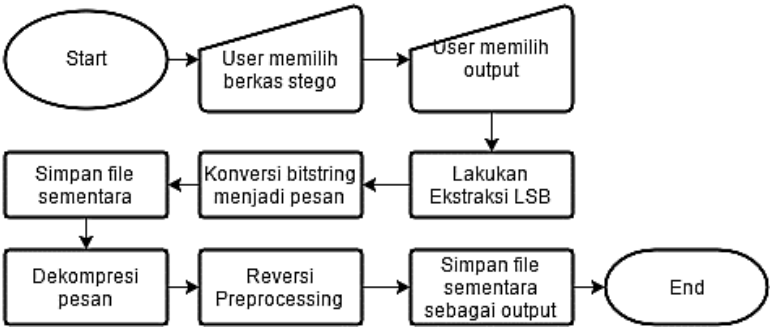
Insertion dengan kolom *bit* sepanjang k dimulai dengan *bit* posisi terkecil hingga posisi *k*. Jika kolom *k* telah dicapai, maka operasi akan dilanjutkan ke *frame* selanjutnya hingga *frame* ke *n*

dan *counter* kolom akan di *reset*. Selanjutnya sisa *frame* pada input akan dituliskan ke berkas output. Berkas output kemudian ditutup.

3.2 Perancangan Tahap *Extraction*

Pada tahap *extraction* terbagi menjadi 3 proses utama, yaitu proses *extraction* LSB, proses dekompresi Adaptive Huffman, dan proses reversi BWT. Semua proses tersebut terhubung melalui 1 fungsi utama yang memiliki diagram alur pada Gambar 3.6 di bawah ini.

Proses dimulai dengan memuat frame pada berkas *cover* dan membaca *header* pada *frame*. Selanjutnya proses *extraction* LSB dilakukan dengan parameter yang telah ditentukan pada *header*. *Bitstring* yang telah terbaca selanjutnya diubah menjadi karakter untuk selanjutnya dimasukkan sebagai input dari proses dekompresi. Hasil dekompresi kemudian masuk ke proses reversi BWT dan menghasilkan berkas pesan sebagai output.

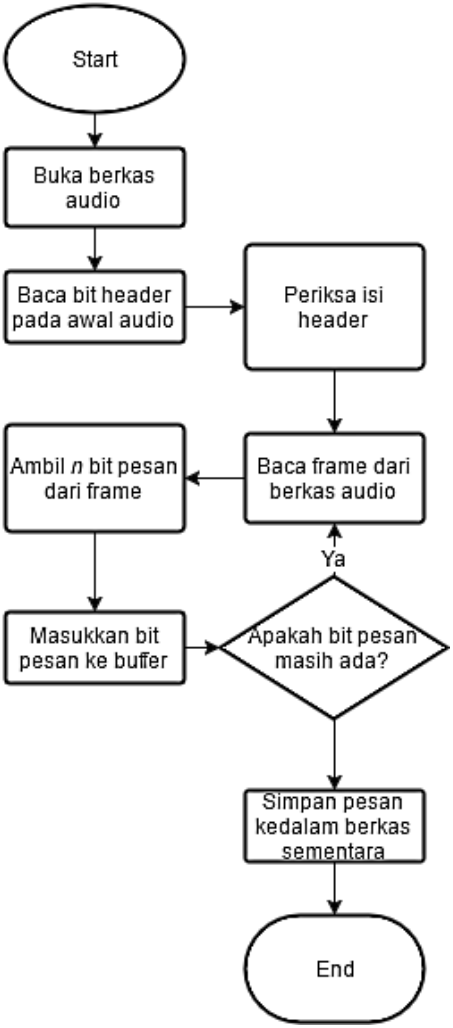


Gambar 3.6 Proses Extraction Global

3.2.1 Proses *Extraction* LSB

Proses *extraction* akan mengambil *bit* pada berkas *cover* untuk selanjutnya disimpan sebagai *bitstring*. Proses ini berjalan sesuai dengan ukuran berkas yang disembunyikan dan parameter

lainnya yang terletak di header. Proses mengikuti alur sesuai dengan Gambar 3.7 berikut.



Gambar 3.7 Proses Extraction LSB

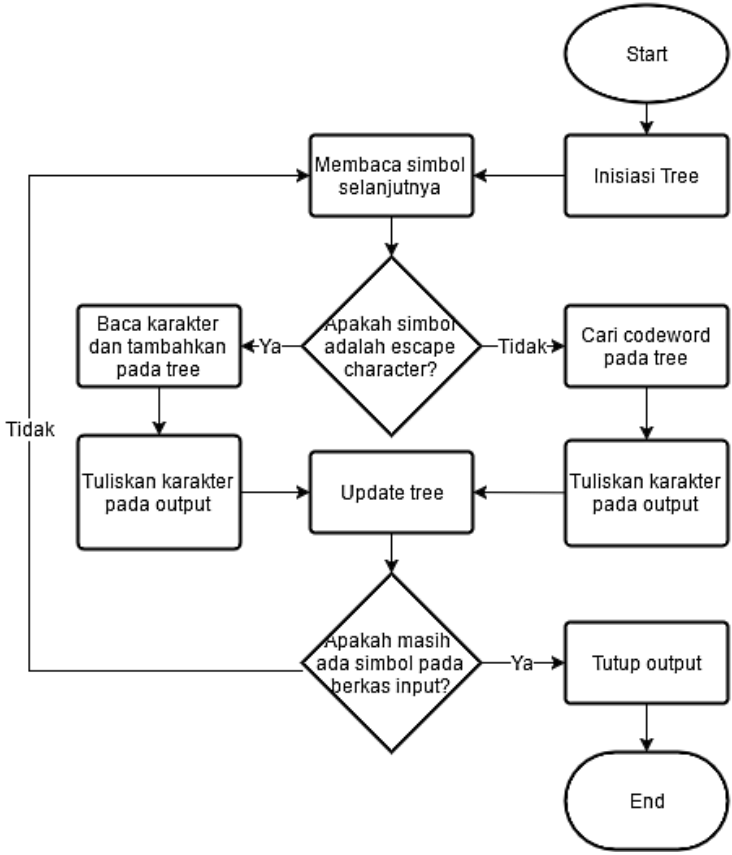
Proses dimulai dengan memuat *frame* dengan jumlah sebanyak *n*/*jumlah_saluran* ke dalam *buffer* berukuran *n*. Selanjutnya lakukan pembacaan *bit* sebanyak *k bit* dimulai dari *bit* posisi terkecil hingga posisi *k*. Apabila setiap elemen *buffer* telah diperiksa, maka *buffer* akan digantikan dengan *frame* baru dari berkas *cover*. Proses ini diulangi hingga jumlah karakter telah mencapai jumlah yang telah ditentukan pada *header*. *Bitstring* pesan yang telah dibaca selanjutnya dikonversi menjadi *bit* dan disimpan dalam berkas sementara untuk selanjutnya digunakan pada proses *extraction* Adaptive Huffman.

Proses pengecekan *bit* pada kolom dilakukan dengan menggunakan operasi bitwise. Jika *bit* pada kolom tertentu bernilai 0, maka tambahkan *bitstring* dengan '0'. Sebaliknya jika *bit* pada kolom bernilai 1, nilai '1' akan ditambahkan pada *bitstring*.

3.2.2 Proses Dekompresi Adaptive Huffman

Proses *dekompresi* Adaptive Huffman mirip dengan proses kompresi dimana pohon huffman diperbaharui pada saat dekompresi berlangsung. Proses *dekompresi* dilakukan melalui proses yang sesuai dengan Gambar 3.8.

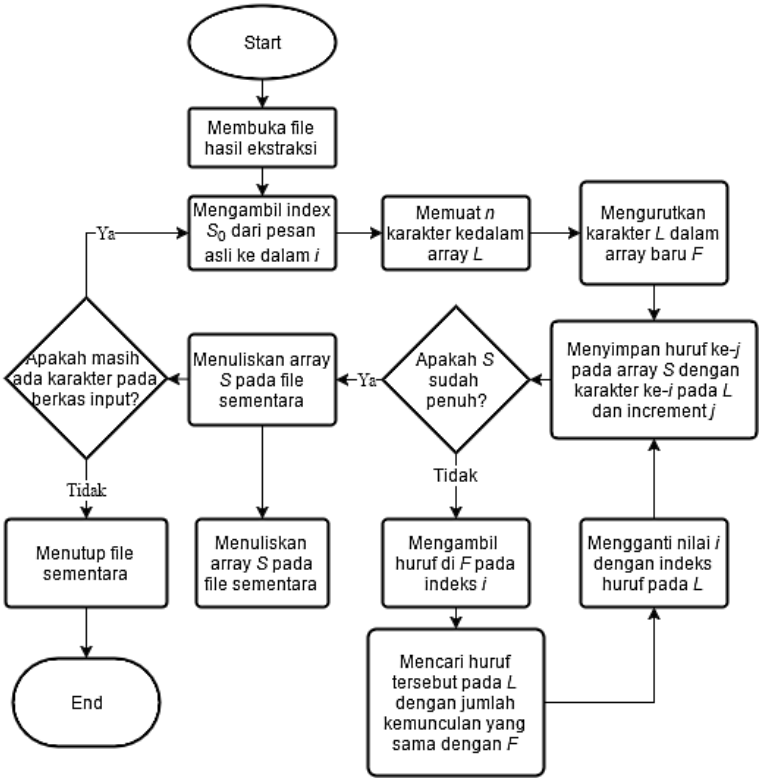
Proses dimulai dengan cara menginisiasi *tree* dengan *leaf* kosong dan memuat simbol pada *buffer*. Selanjutnya, pemeriksaan dilakukan untuk mengetahui apakah simbol adalah *escape character* atau tidak. Pembacaan karakter selanjutnya dilakukan jika simbol tersebut adalah *escape character*. Karakter lalu akan dimasukkan ke dalam *tree* dan dituliskan langsung ke output. Jika simbol tersebut bukan *escape character*, maka karakter hasil *decoding* dari simbol akan dituliskan pada output. *Tree* kemudian akan diperbaharui. Proses akan diulang hingga seluruh simbol pada input telah dibaca. Selanjutnya, *update tree* dengan menambahkan bobot karakter dengan nilai 1. Hasil dari proses ini adalah berkas sementara yang telah di dekompresi. Berkas output kemudian ditutup setelah semua *codeword* pada berkas input telah diproses.



Gambar 3.8 Proses Dekompresi Adaptive Huffman

3.2.3 Proses Reversi BWT

Proses reversi BWT berfungsi untuk melakukan transformasi reversi pada berkas hasil dari proses dekompresi Adaptive Huffman. Tujuannya adalah untuk mendapatkan kembali pesan awal yang telah di transformasi menggunakan transformasi BWT. Reversi ini mengikuti alur sesuai Gambar 3.9



Gambar 3.9 Proses reversi BWT

Proses dimulai dengan membaca S_0(indeks karakter awal) sebagai variabel *i* dan memuat karakter sebanyak *n* pada *array L*. Inisiasi *counter j* sebagai penunjuk nomor karakter hasil. Selanjutnya duplikasi *array L* menjadi *array F* dan urutkan *array* tersebut berdasarkan urutan abjad. Selanjutnya simpan karakter *L*[*i*] pada *S*[*j*] dan tambahkan *j* dengan angka 1. Selanjutnya cari karakter *F*[*i*] pada *L* dengan syarat jumlah kemunculan karakter pada kedua *array* sama. Simpan indeks karakter yang ditemukan pada *i*. Ulangi proses dimulai dari penyimpanan hingga *j=n*. Lakukan hingga semua karakter pada berkas input telah dibaca.

Hasil dari proses ini berupa berkas sementara dengan isi berupa pesan yang telah dikembalikan ke bentuk aslinya.

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dilakukan dengan menggunakan komputer dengan spesifikasi Intel(R) Core(TM) i5-3630QM CPU @ 2.4 GHz dengan memori 8GB. Perangkat lunak yang digunakan dalam proses pengembangan antara lain:

- Sistem Operasi Windows 8 64-*bit*.
- Qt 5.5.0 for Windows 64-*bit*(VS 2013).
- Git 2.6.1.windows.1 untuk kontrol versi kode sumber.
- Microsoft Word 2013 untuk penulisan buku tugas akhir.

4.2 Rincian Implementasi Alur Algoritma *Insertion*

Alur *Insertion* memiliki 3 algoritma utama yang akan diimplementasikan sebagai berikut:

1. Algoritma Transformasi BWT.
2. Algoritma Kompresi Adaptive Huffman.
3. Algoritma *Insertion* LSB.

4.2.1 Algoritma Transformasi BWT

Subbab ini berisi *pseudocode* algoritma Transformasi BWT yang dijelaskan pada *Pseudocode* 4-1.

Masukan	Berkas input messageFile
Keluaran	Berkas output outFile

```
1. blockSize = 4096
2. for (i = 0 to block.length)
3.   baris[i] = i
4. last = malloc(block.length)
5. block = read messageFile with sizeof blockSize
6. blockLength = block.length
7. while(blockLength>0)
8.   qsort(baris,boundedCompare())
9.   for(i = 0 to blockLength)
10.    If(baris[i]==0)
11.      originalCharPosition = i;
12.    last[i]=getLastColumn(baris[i],blockLength);
13.  Write originalCharPosition to outFile;
14.  Write last to outFile
15.  block = read messageFile with sizeof blockSize
16.  blockLength = block.length
17. free block
18. free baris
19. free last
20. return outFile
```

Pseudocode 4-1 Prosedur Transformasi BWT

Pada algoritma ini, terdapat 1 input berupa berkas pesan dan 1 output berupa hasil proses transformasi. Proses dimulai dengan membaca blok berukuran 4096 pada pesan. Selanjutnya dibuat sekuens sebanyak *n* dengan cara membuat *array* berisi indeks awal dari sekuens. Proses dilanjutkan dengan mengurutkan *array* sekuens berdasarkan sekuens yang dihasilkan. Selanjutnya mengambil karakter dengan posisi *n* pada setiap sekuens untuk disimpan pada *array L* dan mencari posisi karakter awal dari blok asli pada *array L* tersebut. Kedua informasi tersebut selanjutnya disimpan dalam output. Proses diulang hingga semua karakter pada pesan telah diproses.

Dalam melakukan pengurutan, terdapat fungsi khusus untuk melakukan *sorting* terhadap *array* baris. Fungsi ini menggunakan nilai pada *array* baris untuk membandingkan

kalimat yang terbentuk dengan cara membaca seluruh huruf dari posisi awal hingga posisi akhir. Jika indeks posisi sudah melebihi batas namun jumlah huruf yang dibandingkan belum mencapai ukuran kalimat, maka posisi kembali ke nilai 0 dan proses dilanjutkan hingga panjang huruf yang telah diurutkan mencapai *n*. *Pseudocode* dari fungsi tersebut dapat dilihat pada *Pseudocode* 4-2.

Masukan	Indeks elemen s1 Indeks elemen s2
Keluaran	Hasil perbandingan k

```
1. blockSize = 4096
2. i = 0
3. c1 = 0
4. c2 = 0
5. for (i = 0 to block.length)
6.    if (s1>=block.length)
7.      s1-=block.length
8.    if (s2>=block.length)
9.      s2-=block.length
10.   c1=block[s1]
11.   c2=block[s2]
12.   if c1>c2 return 1
13.   else if c2>c1 return -1
14.   s1++
15.   s2++
16.return 0
```

Pseudocode 4-2 Algoritma perbandingan

4.2.2 Algoritma Kompresi Adaptive Huffman

Subbab ini berisi *pseudocode* algoritma Kompresi Adaptive Huffman yang dijelaskan pada *Pseudocode* 4-3.

Masukan	Berkas input Berkas output
Keluaran	-

```
1. c = 0
2. END_OF_STREAM = 256
3. Initialize Tree
4. while ((c=getc(input))!= EOF)
5.   EncodeSymbol (Tree, c, output)
6.   Increment character counter to update tree
7. EncodeSymbol (&Tree, END_OF_STREAM, output)
8. return
```

Pseudocode 4-3 Algoritma Kompresi Adaptive Huffman

Algoritma ini menerima 2 input yaitu *pointer* berkas input dan *pointer* berkas output. Selanjutnya selama masih ada karakter di dalam input, fungsi utama yaitu ***EncodeSymbol()*** untuk melakukan proses *encoding* karakter menjadi simbol akan terus dipanggil. Fungsi tersebut dapat dilihat pada *Pseudocode* 4-4 di bawah ini. Selain itu, *update* dilakukan dengan menambahkan *counter* karakter pada *tree* dengan angka 1. Setelah semua karakter telah diproses, maka simbol penanda akhir dari file akan dituliskan.

Masukan	Karakter c Berkas output
Keluaran	-

```
1. code = 0
2. current_bit = 1
3. current_node = leaf[c];
4. if(current_node == -1)
5.   current_node = leaf[Escape_Char]
6. while( current_node!=Root_node)
7.   if((current node & 1)==0)
8.     Code |= current_bit
9.   shl(current_bit)
10.  current_node = nodes[current_node].parent
11.write bits of Code to output)
12.if (leaf[c] == -1)
13.  write bits of c to output)
14.  add_new_node(c)
15.return
```

Pseudocode 4-4 Algoritma Encoding Symbol

Pada fungsi ini, karakter akan diperiksa apakah terdapat pada tree atau tidak. Apabila karakter tersebut memiliki bobot -1, maka karakter tersebut tidak ada di pohon dan escape character akan di-*encode*. Selanjutnya karakter akan dituliskan langsung ke output dan ditambahkan ke pohon. Namun, jika bobot karakter tersebut bukan -1, maka proses penyusunan *bit codeword* akan dilakukan. Proses ini menggunakan operasi *bitwise or* dan *shift left* untuk membuat *codeword* pada setiap kenaikan 1 *node*. Proses diulang hingga *node* sekarang sudah terletak pada *root*.

4.2.3 Algoritma *Insertion* LSB

Subbab ini berisi *pseudocode* algoritma *Insertion* LSB yang dijelaskan pada *Pseudocode* 4-5.

Masukan	Berkas SourceFile Berkas OutputFile String bitstring
Keluaran	-

```
1. soundFile sf = sourceFile
2. soundFile sf_out = job.outFile
3. soundFileInfo sfi = sf_info(sf)
4. k = 0
5. frameCounter = 0
6. bitCounter = 0
7. buffer = malloc(4096)
8. read = read_sf(sf,buffer,4096/sfi.channel)
9. while (read!=0)
10.   while(frameCounter<read*sfi.channel)
11.     while(k<max_bit_column)
12.       if(bitCounter<bitstring.length)
13.         if(bitstring[bitCounter]=='0')
14.           if(buffer[frameCounter]'s k bit !=0)
15.             Invert buffer[frameCounter]'s k bit
16.         else
17.           if(buffer[frameCounter]'s k bit ==0)
18.             Invert buffer[frameCounter]'s k bit
19.         bitCounter++
```

```
20.       else
21.         break
22.       k++
23.     frameCounter++
24.   write_sf(buffer,sf_out,read_
25.   read = read_sf(sf,buffer,4096/sfi.channel)
26. free buffer
27. return
```

Pseudocode 4-5 Algoritma Insertion LSB

Pada proses ini, terdapat 3 input yaitu berkas input, berkas output, dan *string bitstring*. Proses dimulai dengan cara melakukan inisiasi variabel. Selanjutnya data pada berkas suara input dimuat pada *buffer*. Operasi utama dilakukan pada setiap baris dimana *bit* pada baris dimulai dari yang terkecil hingga *bit k* akan diperiksa. Jika *bit* tidak sama dengan *bitstring*, maka operasi *invert* dilakukan. Proses diulang hingga semua *bitstring* telah diproses. Selanjutnya proses akan menuliskan sisa data pada berkas output. Proses ini diakhiri dengan pembersihan memori yang telah dipakai dan menutup berkas yang digunakan.

4.3 Rincian Implementasi Alur Algoritma *Extraction*

Alur *Extraction* memiliki 3 algoritma utama yang akan diimplementasikan sebagai berikut:

1. Algoritma *Extraction* LSB.
2. Algoritma Dekompresi Adaptive Huffman.
3. Algoritma Reversi BWT.

4.3.1 Algoritma Reversi BWT

Subbab ini berisi *pseudocode* algoritma reversi BWT yang dijelaskan pada *Pseudocode* 4-6.

Masukan	Berkas input tempFile
Keluaran	Berkas output outFile

```
1. blockSize = 4096
2. pos=1
3. j=0;
4. for (i=0 to 256)
5.   lastpos[i]=0
6. while(tempFile!=EOF)
7.   pos=read tempFile with sizeof 1
8.   L = read tempFile with sizeof blockSize
9.   memcpy F<-L
10.  sort F in ascending order
11.  j=0
12.  while(j<L.length)
13.    occurenceF=0
14.    occurenceL=0
15.    S[j]=L[pos]
16.    for(k=0 to pos)
17.      if(F[k]==F[pos]
18.        occurenceF++
19.    for(k=0 to i)
20.      if(L[k]==F[pos])
21.        occurenceL++
22.      if(occurenceF==occurenceL)
23.        pos=k
24.        Break
25.    j++
26.  Write S to outfile
27.return outFile
```

Pseudocode 4-6 Algoritma Reversi BWT

Pada proses ini, dilakukan pembacaan dari berkas hasil proses ekstraksi LSB pada input tempFile. Terdapat 2 jenis informasi yang perlu dibaca secara berurutan, yaitu nomor indeks karakter awal yang dimasukkan ke dalam variabel *pos* dan huruf yang telah di transformasi yang dimasukkan ke dalam *array L*. Lakukan duplikasi *L* pada variabel *F* dan urutkan *F* berdasarkan abjad. Proses utama yaitu penyusunan *array S* dimulai. Proses dimulai menuliskan karakter pada *L* dengan posisi *pos* pada *array* hasil *S*. Selanjutnya ambil karakter pada *F* yang memiliki posisi *pos* dan hitung jumlah kemunculannya hingga posisi karakter tersebut. Cari karakter tersebut pada *L* dengan jumlah kemunculan yang sama. Simpan posisi karakter yang ditemukan pada *pos*.

Proses utama akan dilakukan hingga jumlah karakter yang ditulis sebanyak ukuran blok. Semua proses diulangi hingga tidak ada karakter yang belum diproses.

4.3.2 Algoritma Dekompresi Adaptive Huffman

Subbab ini berisi *pseudocode* algoritma Dekompresi Adaptive Huffman yang dijelaskan pada *Pseudocode* 4-7.

Masukan	Berkas input Berkas output
Keluaran	-

```
9.  c = 0
10. END_OF_STREAM = 256
11. Initialize Tree
12.  while ((c=DecodeSymbol( &Tree, input))!= EOF)
13.   write c to output
14.   Increment character counter to update tree
15. return
```

Pseudocode 4-7 Algoritma Dekompresi Adaptive Huffman

Proses ini menerima 2 jenis berkas yaitu input dan output sebagai input. Proses dimulai dengan menginisiasi pohon dan melakukan *decoding* terhadap karakter. Selama hasil decoding bukan karakter penutup, Maka karakter akan dituliskan pada output. Untuk melakukan *decoding*, fungsi ini menggunakan fungsi utama berupa ***DecodeSymbol()*** untuk melakukan *decoding* terhadap simbol yang ada. Selanjutnya *tree* akan di-*update* dengan melakukan penambahan terhadap *counter* karakter dengan angka 1. Fungsi ***DecodeSymbol()*** dijelaskan pada *Pseudocode* 4-8 di bawah ini.

Masukan	Berkas input
Keluaran	Karakter C

```
1.  current_node = Root_Node
2.  c = 0
3.  while(node[current_node].child_is_leaf != true)
4.    current_node=node[current_node].child;
5.    currentNode+=GetBit(input);
6.  c = node[current_node].child;
7.  if( c == Escape_Character)
8.    c = getbit(input)
9.    add_new_node(c)
10. return c
```

Pseudocode 4-8 Algoritma DecodeSymbol

Proses *decoding* dimulai dengan melakukan pengecekan tree berdasarkan *bit* pada *codeword*. Pengecekan dilakukan hingga akhirnya mencapai *parent* dari *node leaf* karakter yang sesuai dengan *codeword*. Selanjutnya karakter dikembalikan ke fungsi utama. Jika karakter yang dihasilkan adalah *escape character*, maka pembacaan berkas akan dilakukan untuk mendapatkan karakter yang tidak terkompresi dan menambahkannya pada pohon.

4.3.3 Algoritma Extraction LSB

Subbab ini berisi *pseudocode* algoritma *Extraction* LSB yang dijelaskan pada *Pseudocode* 4-9.

Masukan	Berkas input Berkas output Integer Msg_Size
Keluaran	-

```
1.  soundFile sf = sourceFile
2.  soundFileInfo sfi = sf_info(sf)
3.  k = 0;
4.  bitstring = ''
5.  message = ''
6.  buffer = malloc(4096)
7.  frameCounter = 0;
8.  bitCounter = 0;
9.  read = read_sf(sf,buffer,4096/sfi.channel)
10. while (read!=0)
```

```
11.   while(frameCounter<read*sfi.channel && !done)
12.     while(k<max_bit_column)
13.       if(bitCounter<Msg_Size)
14.         if(buffer[frameCounter]'s k bit == 0)
15.           Append(bitstring,'0')
16.         else
17.           Append(bitsring,'1')
18.         bitCounter++
19.       else
20.         break
21.       k++
22.     frameCounter++
23.   read = read_sf(sf,buffer,4096/sfi.channel)
24. message = convertBitString(bitstring)
25. write message to output
26. free buffer
27. Return
```

Pseudocode 4-9 Implementasi Extraction LSB

Proses ini menerima 3 jenis input berupa berkas input, berkas output, dan ukuran pesan. Inisiasi variabel dilakukan pada awal proses. Selanjutnya, sebanyak 4096/*jumlah_saluran frame* dimuat pada *buffer*. Proses dilanjutkan dengan membaca sebanyak *k bit* pada setiap elemen *buffer* dimulai dari *bit* terkecil hingga *bit* pada posisi *k*. Jika *bit* pada *frame* bernilai 0, maka *bitstring* akan ditambahkan karakter '0'. Dan jika *bit* bernilai 1, maka *bitstring* akan ditambahkan karakter '1'. Jika semua *buffer* telah diproses, data baru dimuat pada *buffer*. Proses diulang hingga mencapai batas pada ukuran pesan. Selanjutnya konversi *bit* akan dilakukan dengan hasil berupa data hasil ekstraksi yang sebenarnya. Data tersebut selanjutnya disimpan pada berkas output.

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan uji coba yang dilakukan pada aplikasi yang telah dikerjakan serta analisa dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi lingkungan uji coba, skenario uji coba yang meliputi uji kebenaran dan uji kinerja serta analisa setiap pengujian.

5.1 Lingkungan Uji Coba

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji program Steganografi LSB dan Kompresi Adaptive Huffman BWT. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

1. Perangkat keras
    a. Prosesor: Intel® Core™ i7-3630QM CPU @ 2.30GHz .
    b. *Memory* (RAM): 8,00 GB.
    c. Tipe sistem: Sistem Operasi 64-bit.
    d. *Headphone*: Sony® MDR-EX15 *Stereo Headphone*.
2. Perangkat lunak
    a. Sistem operasi: Windows 8 .
    b. Pemutar berkas: *Media Player Classic* versi 1.7.7.128.

5.2 Dataset Uji Coba

Pada tugas akhir ini, pengujian akan dilakukan dengan menggunakan 2 jenis dataset sebagai masukkan program. Berkas yang digunakan sebagai berkas pesan adalah berkas teks dengan panjang beragam yang bersumber dari dataset Calgary Corpus [10], berkas ini selanjutnya akan dimasukkan ke dalam berkas *cover*. Berkas yang digunakan sebagai berkas *cover* adalah 3 jenis berkas suara yang selanjutnya dikonversi sehingga memiliki 3 resolusi *bit* yang berbeda. Ukuran dari berkas yang akan digunakan dapat dilihat pada Tabel 5-1 di bawah ini.

Tabel 5-1 Ukuran berkas pada dataset

No	Nama	Jenis	Ukuran (byte)
1	Bib.txt	Teks	111.261
2	Geo.txt	Teks	102.400
3	Obj1.txt	Teks	21.504
4	paper1.txt	Teks	53.161
5	Paper2.txt	Teks	82.199
6	Paper3.txt	Teks	46.526
7	Paper4.txt	Teks	13.286
8	Paper5.txt	Teks	11.954
9	Paper6.txt	Teks	38.105
10	Progc.txt	Teks	39.611
11	Progl.txt	Teks	71.646
12	Progp.txt	Teks	49.379
13	Violin8.wav	Audio	5.128.236
14	Violin16.wav	Audio	10.256.428
15	Violin24.wav	Audio	15.384.620
16	Other8.wav	Audio	4.836.140
17	Other16.wav	Audio	9.672.236
18	Other24.wav	Audio	14.508.332
19	Wildlife8.wav	Audio	2.639.948
20	Wildlife16.wav	Audio	5.279.532
21	Wildlife24.wav	Audio	7.919.116

Panjang berkas teks yang berbeda-beda digunakan untuk menganalisis rasio kompresi yang dihasilkan dan membandingkannya dengan performa kompresi yang telah ada sebelumnya. Sementara resolusi berkas *cover* yang berbeda-beda dan jumlah *bit* yang diubah pada saat proses pengujian berlangsung digunakan untuk melihat pengaruhnya terhadap nilai PSNR dan SNR yang dihasilkan serta waktu pengujian yang digunakan.

5.3 Skenario dan Evaluasi Pengujian

Uji coba ini dilakukan untuk menguji apakah fungsionalitas program telah diimplementasikan dengan benar dan berjalan sesuai dengan rancangan yang telah dibuat sebelumnya. Uji coba akan didasarkan pada beberapa skenario untuk menguji kinerja aplikasi. Tiap uji coba akan menghasilkan PSNR, SNR, rasio kompresi eksekusi, dan waktu pengujian.

Skenario pengujian terdiri dari 5 skenario pengujian yaitu:

1. Skenario perhitungan nilai PSNR dan SNR untuk proses *insertion* berkas teks terkompresi ke dalam 3 jenis berkas suara dengan 3 jenis resolusi dan 3 jenis *bit* yang dipakai.
2. Skenario perhitungan nilai PSNR dan SNR untuk proses *insertion* berkas teks tidak terkompresi ke dalam 3 jenis berkas suara dengan 3 jenis resolusi dan 3 jenis *bit* yang dipakai.
3. Skenario perhitungan nilai rasio kompresi untuk proses *insertion* 12 jenis berkas teks ke dalam salah satu jenis berkas *cover*.
4. Skenario perhitungan peforma aplikasi dalam melakukan proses *insertion* berdasarkan waktu eksekusi yang dibutuhkan.
5. Skenario perhitungan subjektif dalam menentukan kualitas berkas *stego* yang baik dimana terdapat 4 responden yang akan mendengarkan 4 berkas suara dan menentukan kualitas berkas terbaik.

5.3.1 Skenario Uji Coba 1

Skenario uji coba 1 adalah dengan melakukan perhitungan perhitungan nilai PSNR dan SNR untuk proses *insertion* berkas teks terkompresi ke dalam 3 jenis berkas suara dengan 3 jenis resolusi berkas dan 3 jenis jumlah yang dipakai.

Jenis berkas suara yang digunakan dapat dilihat sebagai berikut:

1. Jenis 1 yaitu suara biola dengan panjang 58 detik (violin*.wav).

2. Jenis 2 yaitu suara kotak musik dengan panjang 54 detik (Other*.wav).
3. Jenis 3 yaitu suara gitar dengan panjang 30 detik (Wildlife*wav).

Semua jenis berkas memiliki 3 jenis resolusi yaitu 8, 16, dan 24 *bit*, sehingga terdapat total 9 berkas yang akan dijadikan sebagai berkas *cover*. Spesifikasi dari setiap berkas yang akan digunakan dapat dilihat pada Tabel 5-2.

Tabel 5-2 Spesifikasi berkas input pada proses insertion

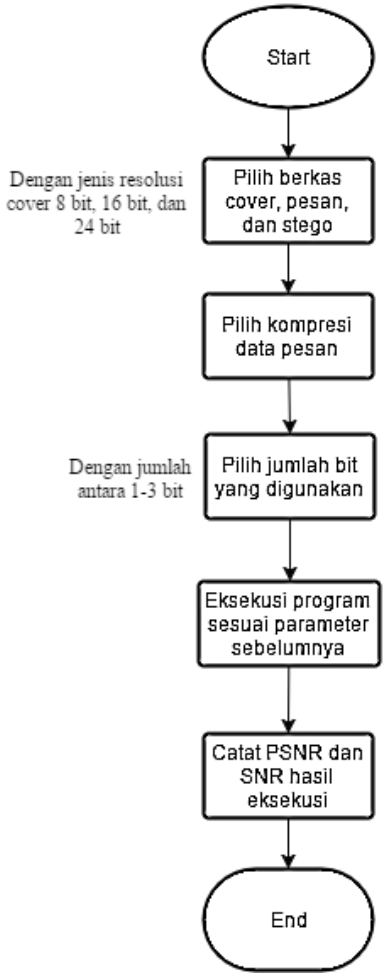
No	Nama	Jenis	Durasi (s)	Resolusi (bit)	Jumlah Frame
1	Bib.txt	Teks	-	-	-
2	Violin8.wav	Audio	58	8	2.564.096
3	Violin16.wav	Audio	58	16	2.564.096
4	Violin24.wav	Audio	58	24	2.564.096
5	Other8.wav	Audio	54	8	2.418.048
6	Other16.wav	Audio	54	16	2.418.048
7	Other24.wav	Audio	54	24	2.418.048
8	Wildlife8.wav	Audio	30	8	1.319.792
9	Wildlife16.wav	Audio	30	16	1.319.792
10	Wildlife24.wav	Audio	30	24	1.319.792

Jenis jumlah *bit* yang digunakan pada pengujian ini merentang dari 1 *bit* hingga 3 *bit*. *Bit* yang digunakan nantinya dimulai dari *bit* terkecil hingga *bit* k dengan k adalah batas jumlah *bit* yang ditentukan.

Penggunaan beragam jenis berkas dan beragam jumlah bit digunakan menganalisis hubungan antara kedua komponen pengujian tersebut dengan tingkat PSNR dan SNR yang dihasilkan oleh proses tersebut. Selain itu, dapat ditentukan juga jenis berkas apa dan berapa banyak *bit* yang cocok digunakan pada steganografi.

Sesuai dengan Gambar 5.1, alur jalannya program dimulai dengan memilih berkas *cover*, berkas pesan dan berkas *stego* yang

akan digunakan untuk eksekusi. Kemudian proses dilanjutkan dengan memilih pilihan untuk menggunakan kompresi. Selanjutnya tentukan panjang *bit* yang akan digunakan. Lakukan proses insertion dan catat nilai PSNR dan SNR hasil pengujian.



Gambar 5.1 Skenario Uji Coba 1

5.3.2 Evaluasi Uji Coba 1

Setelah dilakukan uji coba sesuai dengan skenario uji coba 1, didapatkan nilai PSNR dan SNR berbeda-beda untuk setiap jenis berkas *cover*. Perbandingan nilai statistik hasil eksekusi akan dijelaskan pada setiap subbab berikut ini.

5.3.2.1 Evaluasi Uji Coba 1 Pada Berkas Jenis 1

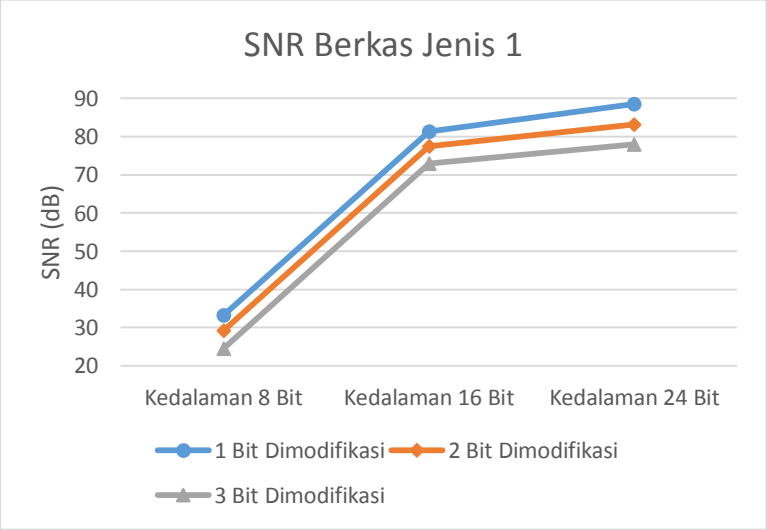
Hasil Uji Coba 1 pada berkas jenis 1 dapat dilihat pada Gambar 5.2 dan Gambar 5.3. Pada Gambar 5.2 dapat dilihat perbedaan nilai PSNR pada setiap jenis resolusi dan jumlah *bit* yang digunakan. Pada Gambar 5.3 dapat dilihat perbedaan nilai SNR pada setiap jenis resolusi dan jumlah *bit* yang digunakan. Perbandingan statistik berkas *stego* yang dihasilkan dari proses ini dapat dilihat pada Tabel 5-3.

Berdasarkan tabel 5-3, dapat dilihat bahwa pada berkas jenis ini PSNR dan SNR pada resolusi 24 bit dengan batas penggunaan bit berjumlah 1 memiliki nilai terbaik dibandingkan dengan jenis berkas dan jenis batas bit lainnya.

Tabel 5-3 Perbandingan statistik hasil proses berkas jenis 1

No	Resolusi Berkas	Batas Bit	Frame Terpakai	PSNR (dB)	SNR (dB)
1	8	1	580.125	54,5859	33,2633
2	8	2	290.081	50,5237	29,2012
3	8	3	193.400	45,9092	24,5867
4	16	1	580.125	96,7553	81,4070
5	16	2	290.081	92,7599	77,4116
6	16	3	193.400	88,3010	72,9527
7	24	1	580.125	145,2390	88,5751
8	24	2	290.081	139,8850	83,2214
9	24	3	193.400	134,6740	78,0099

Gambar 5.2 Grafik PSNR berkas jenis 1



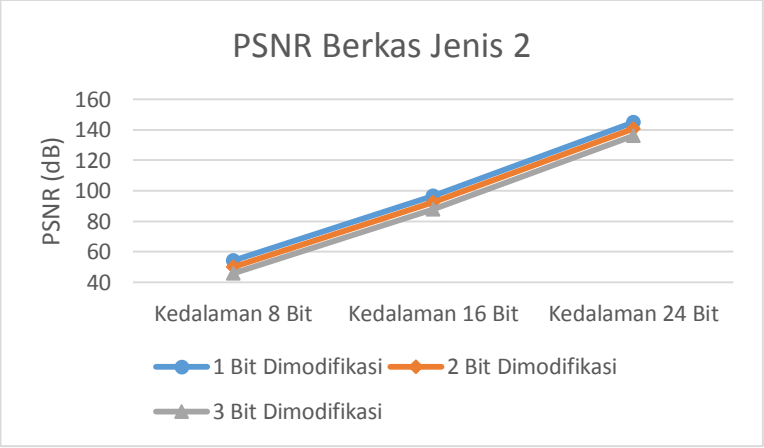
Gambar 5.3 Grafik SNR berkas jenis 1

5.3.2.2 Evaluasi Uji Coba 1 Pada Berkas Jenis 2

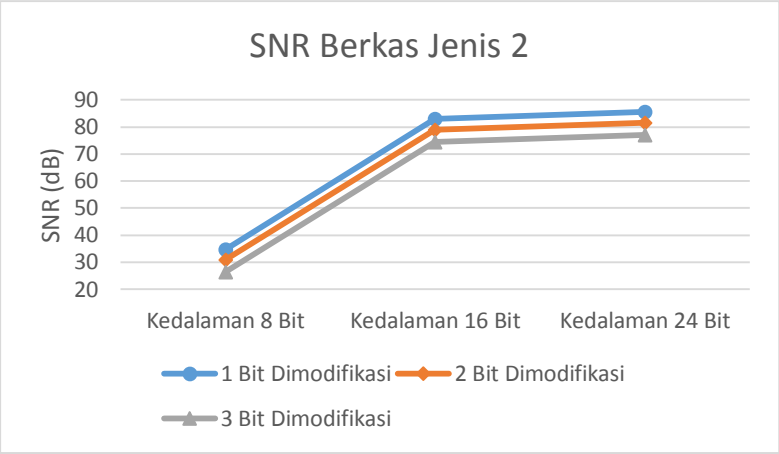
Hasil Uji Coba 1 pada berkas jenis 2 dapat dilihat pada Gambar 5.4 untuk PSNR dan Gambar 5.5 untuk SNR. Perbandingan statistik berkas *stego* dapat dilihat pada Tabel 5-4.

Tabel 5-4 Perbandingan statistik hasil proses berkas jenis 2

No	Resolusi Berkas	Batas Bit	Frame Terpakai	PSNR (dB)	SNR (dB)
1	8	1	580.125	54,3290	34,8198
2	8	2	290.081	50,3162	30,8070
3	8	3	193.400	45,8248	26,3156
4	16	1	580.125	96,5124	82,9820
5	16	2	290.081	92,5171	78,9867
6	16	3	193.400	88,0409	74,5105
7	24	1	580.125	144,6700	85,6131
8	24	2	290.081	140,6790	81,6222
9	24	3	193.400	136,2150	77,1588



Gambar 5.4 Grafik PSNR berkas jenis 2



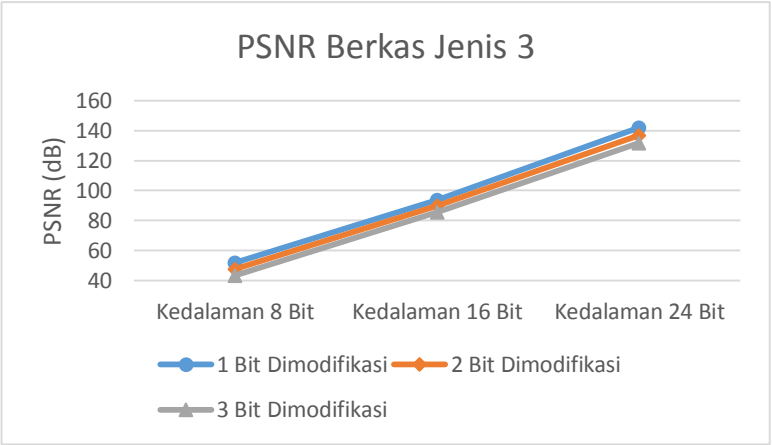
Gambar 5.5 Grafik SNR berkas jenis 2

5.3.2.3 Evaluasi Uji Coba 1 Pada Berkas Jenis 3

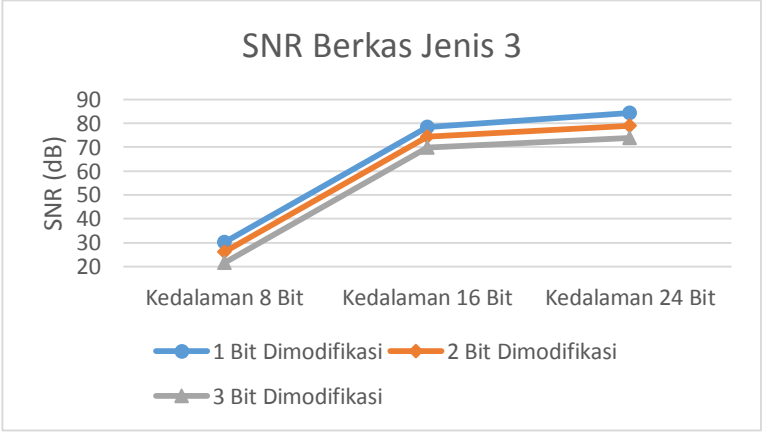
Hasil Uji Coba 1 pada berkas jenis 3 dapat dilihat pada Gambar 5.6 untuk PSNR dan Gambar 5.7 untuk SNR. Perbandingan statistik berkas *stego* dapat dilihat pada Tabel 5-5.

Tabel 5-5 Perbandingan statistik hasil proses berkas jenis 3

No	Resolusi Berkas	Batas Bit	Frame Terpakai	PSNR (dB)	SNR (dB)
1	8	1	580.125	51,6994	30,1513
2	8	2	290.081	47,7083	26,1602
3	8	3	193.400	43,3053	21,7572
4	16	1	580.125	93,8909	78,3167
5	16	2	290.081	89,8752	74,3010
6	16	3	193.400	85,4274	69,8533
7	24	1	580.125	142,3550	84,4206
8	24	2	290.081	137,0010	79,0669
9	24	3	193.400	131,7890	73,8553



Gambar 5.6 Grafik PSNR berkas jenis 3



Gambar 5.7 Grafik PSNR berkas jenis 3

5.3.2.4 Analisis Uji Coba 1

Berdasarkan hasil dari setiap uji coba yang dilakukan pada berkas di atas, dapat dilihat bahwa terdapat kenaikan nilai PSNR yang signifikan antara setiap resolusi. Namun perbedaan SNR pada berkas dengan resolusi 16 *bit* dan 24 *bit* terlihat tidak terlalu

signifikan meskipun kenaikan nilai dari resolusi 8 *bit* dan 16 *bit* terlihat jelas. Hal ini disebabkan oleh jangkauan nilai pada resolusi 16 *bit* dan 24 *bit* sangat tinggi, sehingga penurunan kualitas akibat perubahan pada *bit* LSB menjadi tidak signifikan.

Pada tabel terlihat bahwa penggunaan 1 *bit* memiliki nilai PSNR dan SNR yang lebih tinggi dengan nilai tertinggi pada resolusi 24 *bit*. Namun, penggunaan 1 *bit* pada proses membutuhkan jumlah *frame* lebih banyak untuk menyembunyikan pesan. Hal ini disebabkan oleh proses yang hanya menggunakan 1 *bit* dari setiap *frame* untuk menyisipkan berkas pesan. Sementara pada penggunaan 2 *bit* dan 3 *bit* terjadi penurunan pada nilai PSNR dan SNR dengan jangkauan 1-15 dB, namun menggunakan jumlah *frame* yang lebih sedikit dibandingkan dengan hanya menggunakan 1 *bit*.

Dari pengamatan di atas dapat diambil kesimpulan bahwa pada setiap berkas dengan resolusi tinggi memiliki nilai PSNR yang lebih baik dibandingkan dengan berkas yang memiliki resolusi rendah. Selain itu, proses yang dibatasi hanya pada 1 *bit* saja menghasilkan nilai PSNR dan SNR yang lebih baik dibandingkan pada proses yang dapat menggunakan 2-3 *bit* LSB dari *frame*. Namun, nilai jumlah *frame* yang digunakan pada proses yang dibatasi hanya pada 1 *bit* tergolong tinggi sehingga lebih cocok digunakan untuk berkas dengan jumlah *frame* yang banyak. Pada proses dengan menggunakan *bit* LSB lebih dari 1, nilai jumlah *frame* yang digunakan menurun sesuai dengan rumus berikut:

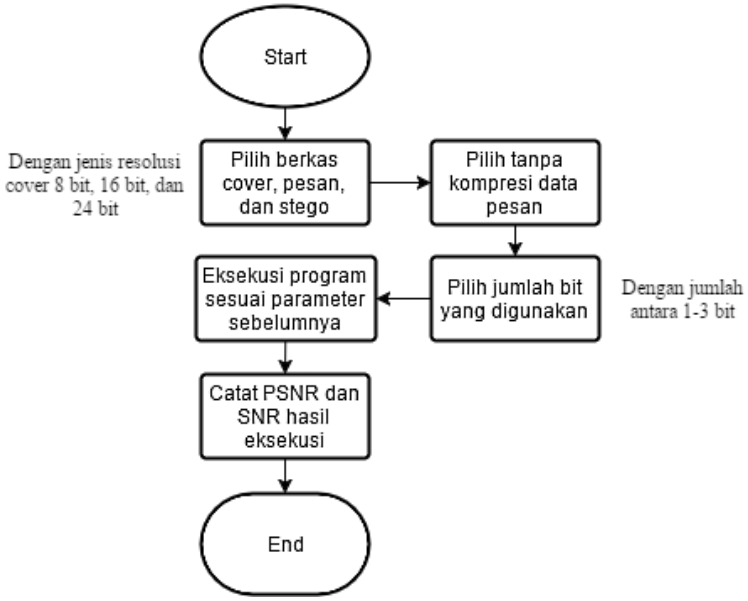
$$Frame\ terpakai = Ceiling\left(\frac{Frame\ terpakai\ pada\ LSB\ 1}{Bit\ yang\ digunakan}\right)$$

Penggunaan 2-3 *bit* LSB dapat dijadikan alternatif jika berkas *cover* yang digunakan pada saat proses steganografi dilakukan tidak memiliki jumlah *frame* yang besar.

5.3.3 Skenario Uji Coba 2

Skenario uji coba 2 adalah dengan melakukan perhitungan perhitungan nilai PSNR dan SNR untuk proses *insertion* berkas teks yang tidak dikompresi ke dalam berkas *cover*. Dimana penggunaan berkas teks yang tidak terkompresi digunakan untuk menganalisis tingkat keberhasilan proses kompresi dalam menaikkan nilai PSNR dan SNR dari berkas *stego*. Selain kompresi, tidak ada parameter yang berbeda dengan uji coba 1 dimana terdapat 9 jenis berkas suara dan 3 jenis penggunaan *bit*.

Sesuai dengan Gambar 5.8, alur dimulai dengan memilih berkas *cover*, berkas pesan dan berkas *stego* yang akan digunakan, kemudian memilih pilihan untuk tidak menggunakan kompresi. Selanjutnya, tentukan panjang *bit* yang akan digunakan. Lakukan proses *insertion* dan catat nilai PSNR dan SNR hasil pengujian.



Gambar 5.8 Skenario Uji Coba 1

5.3.4 Evaluasi Uji Coba 2

Setelah dilakukan uji coba sesuai skenario uji coba 2, didapatkan nilai PSNR dan SNR berbeda-beda untuk setiap jenis berkas *cover*. Perbandingan nilai statistik hasil eksekusi dibagi sesuai jenis berkas *cov*er dan akan dijelaskan pada setiap subbab berikut ini.

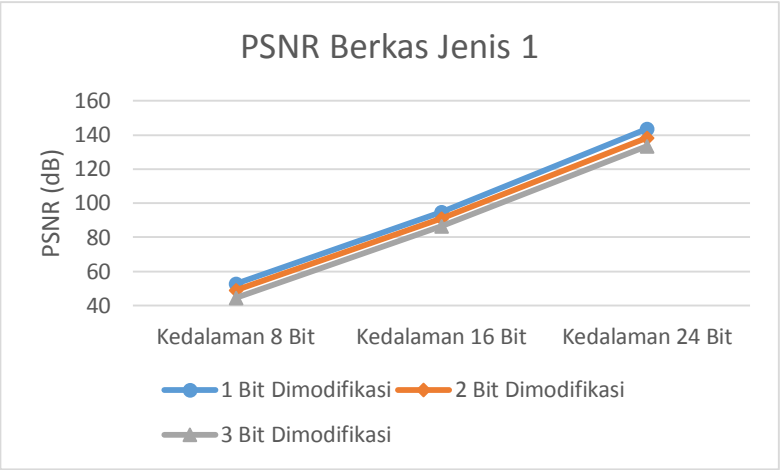
5.3.4.1 Evaluasi Uji Coba 2 Pada Berkas Jenis 1

Hasil Uji Coba 2 pada berkas jenis 1 dengan resolusi dan jumlah bit yang berbeda dapat dilihat pada Gambar 5.9 untuk PSNR dan 5.10 untuk SNR. Rincian perbandingan statistik berkas *stego* hasil dari proses dapat dilihat pada Tabel 5-6.

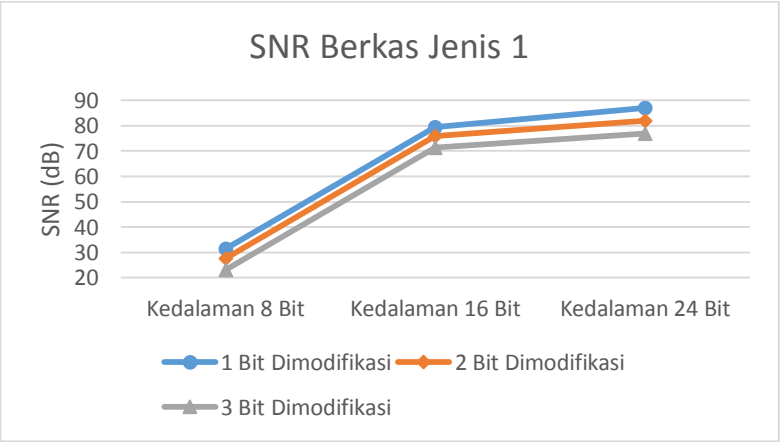
Berdasarkan tabel 5-4, dapat dilihat bahwa pada berkas jenis ini PSNR dan SNR pada resolusi 24 bit dengan batas penggunaan bit berjumlah 1 memiliki nilai terbaik dibandingkan dengan jenis berkas lainnya. Namun, nilai terbaik yang dihasilkan lebih rendah dibandingkan dengan nilai terbaik dari Uji Coba 1.

Tabel 5-6 Perbandingan statistik hasil proses berkas jenis 1

No	Resolusi Berkas	Batas Bit	Frame Terpakai	PSNR (dB)	SNR (dB)
1	8	1	890.125	52,7247	31,4022
2	8	2	445.081	48,9603	27,6377
3	8	3	296.733	44,2932	22,9706
4	16	1	890.125	94,9038	79,5555
5	16	2	445.081	91,1783	75,8299
6	16	3	296.733	86,6071	71,2587
7	24	1	890.125	143,7360	87,0718
8	24	2	445.081	138,5480	81,8840
9	24	3	296.733	133,4830	76,8196



Gambar 5.9 Grafik PSNR berkas jenis 1



Gambar 5.10 Grafik SNR berkas jenis 1

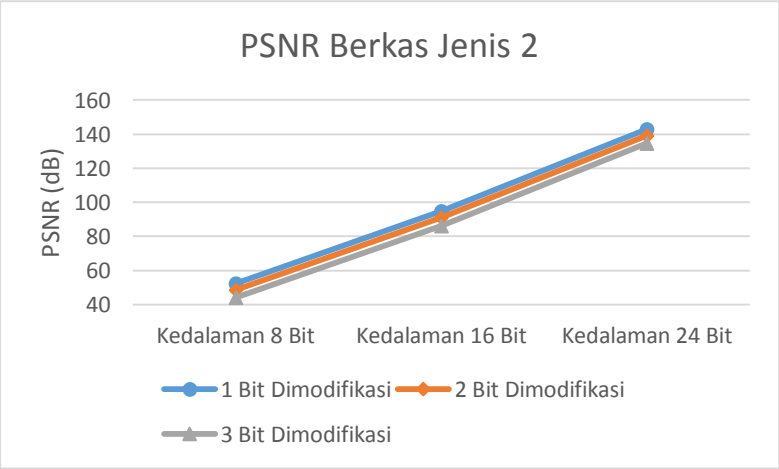
5.3.4.2 Evaluasi Uji Coba 2 Pada Berkas Jenis 2

Hasil Uji Coba 2 pada berkas jenis 2 dapat dilihat pada Gambar 5.11 dan 5.12. Pada Gambar 5.11 dapat dilihat perbedaan nilai PSNR pada setiap jenis resolusi dan jumlah *bit* yang

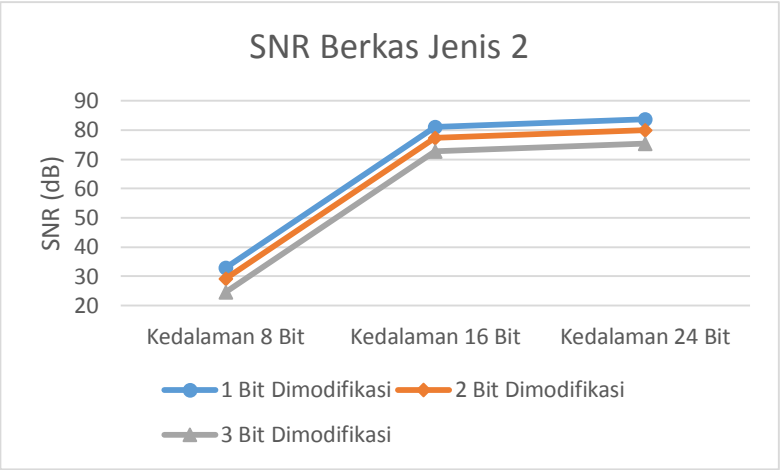
digunakan. Sementara perbedaan SNR dapat dilihat pada Gambar 5.12. Perbandingan statistik berkas *stego* hasil dari proses dapat dilihat pada Tabel 5-7.

Tabel 5-7 Perbandingan statistik hasil proses berkas jenis 2

No	Resolusi Berkas	Batas Bit	Frame Terpakai	PSNR (dB)	SNR (dB)
1	8	1	890.125	52,4732	32,9640
2	8	2	445.081	48,7485	29,2392
3	8	3	296.733	44,1541	24,6449
4	16	1	890.125	94,6497	81,1193
5	16	2	445.081	90,9165	77,3861
6	16	3	296.733	86,3436	72,8132
7	24	1	890.125	142,8140	83,7575
8	24	2	445.081	139,0900	80,0330
9	24	3	296.733	134,5170	75,4604



Gambar 5.11 Grafik PSNR berkas jenis 2



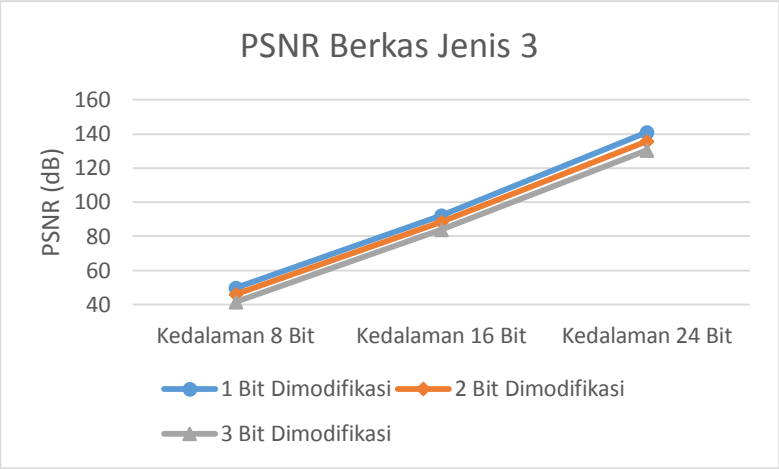
Gambar 5.12 Grafik SNR berkas jenis 2

5.3.4.3 Evaluasi Uji Coba 2 Pada Berkas Jenis 3

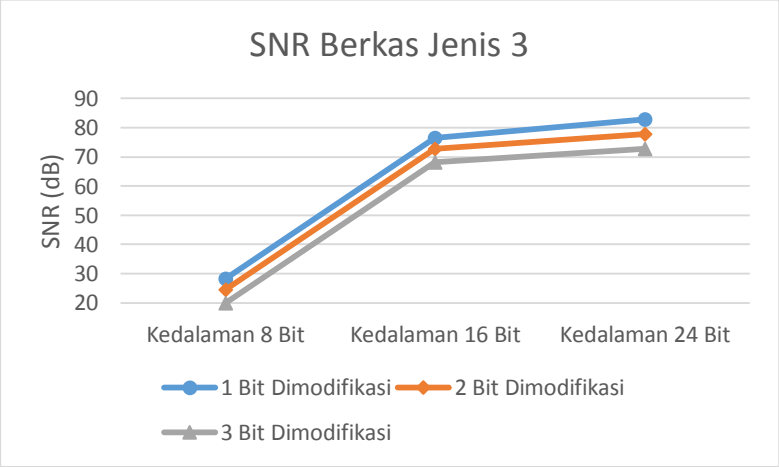
Hasil Uji Coba 2 pada berkas jenis 3 dapat dilihat pada Gambar 5.13 untuk PSNR dan Gambar 5.14 untuk SNR. Perbandingan statistik berkas *stego* dapat dilihat pada Tabel 5-8.

Tabel 5-8 Perbandingan statistik hasil proses berkas jenis 3

No	Resolusi Berkas	Batas Bit	Frame Terpakai	PSNR (dB)	SNR (dB)
1	8	1	890.125	49,8386	28,2905
2	8	2	445.081	46,1136	24,5655
3	8	3	296.733	41,5780	20,0298
4	16	1	890.125	92,0172	76,4430
5	16	2	445.081	88,2889	72,7147
6	16	3	296.733	83,7283	68,1541
7	24	1	890.125	140,8510	82,9173
8	24	2	445.081	135,6630	77,7295
9	24	3	296.733	130,5990	72,6651



Gambar 5.13 Grafik PSNR berkas jenis 3



Gambar 5.14 Grafik SNR berkas jenis 3

Sesuai dengan hasil kedua jenis berkas *cover* sebelumnya, terjadi peningkatan jumlah *frame* yand digunakan dan penurunan nilai PSNR dan SNR jika dibandingkan pada hasil uji coba 1. Hal ini disebabkan oleh meningkatnya jumlah *bit* yang akan disisipkan

karena tidak adanya proses kompresi yang dapat mengurangi ukuran berkas pesan.

5.3.4.4 Analisis Uji Coba 2

Pada hasil uji coba 2, didapatkan bahwa penggunaan berkas dengan resolusi tinggi tetap menghasilkan nilai PSNR dan SNR yang lebih baik dibandingkan berkas dengan resolusi rendah. Sehingga dapat disimpulkan bahwa steganografi akan menghasilkan nilai PSNR dan SNR yang baik jika digunakan pada berkas *cover* dengan resolusi tinggi.

Berdasarkan hasil uji coba 1 dan uji coba 2, didapatkan bahwa terdapat kenaikan nilai PSNR dan SNR pada berkas hasil uji coba 1 dimana berkas pesan melewati proses kompresi terlebih dahulu sebelum di sisipkan pada berkas *cover*. Hal ini menyebabkan berkurangnya jumlah *bit* yang harus disisipkan pada berkas *cover*. Sehingga jumlah penggunaan *frame* pun ikut berkurang.

5.3.5 Skenario Uji Coba 3

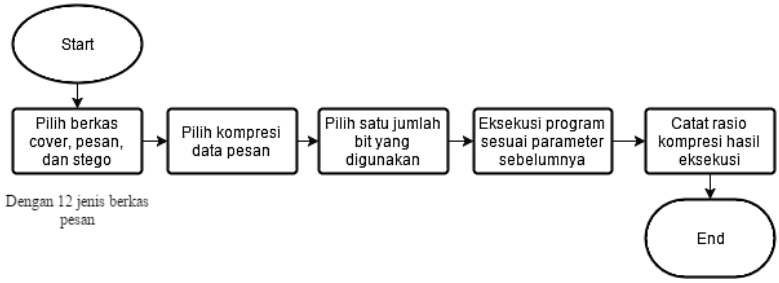
Skenario uji coba 3 adalah melakukan perhitungan nilai rasio kompresi untuk proses kompresi pada saat proses insertion dilakukan. Dataset yang digunakan adalah 12 jenis berkas teks dengan panjang yang berbeda-beda yang bersumber dari Calgary *Corpus*. Spesifikasi dataset dapat dilihat pada Tabel 5-9 berikut ini.

Tabel 5-9 Spesifikasi data set teks

Nama Berkas	Ukuran (Byte)	Deskripsi
Bib	111.261	Bibliografi
Geo	102.400	Data Geologi
Obj1	21.504	Program objek VAX
paper1	53.161	Jurnal Teknis

Nama Berkas	Ukuran (Byte)	Deskripsi
Paper2	82.199	Jurnal Teknis
Paper3	46.526	Jurnal Teknis
Paper4	13.286	Jurnal Teknis
Paper5	11.954	Jurnal Teknis
Paper6	38.105	Jurnal Teknis
Progc	39.611	Kode Sumber dalam "C"
Progl	71.646	Kode Sumber dalam "Pascal"
Progp	49.379	Teks Bahasa Inggris

Penggunaan beragam jenis berkas teks digunakan untuk melihat keberhasilan dari teknik kompresi yang dipakai terhadap teknik kompresi yang telah ada sebelumnya. Keberhasilan dilihat dari perbandingan antara rasio kompresi teknik kompresi yang digunakan dengan rasio teknik kompresi yang telah ada sebelumnya. Alur proses pengujian dapat dilihat pada Gambar 5.15.



Gambar 5.15 Skenario uji coba 3

5.3.6 Evaluasi Uji Coba 3

Hasil uji coba 3 pada 12 jenis berkas teks dapat dilihat pada Tabel 5-10 di bawah ini. Dapat dilihat bahwa ukuran dari setiap berkas berkurang dengan rasio yang berbeda-beda. Sementara hasil

pebandingan rasio kompresi dapat dilihat pada Gambar 5.16 dan Tabel 5-11. Dimana kompresi 1 adalah *arithmatic coding*, kompresi 2 adalah Huffman+BWT, dan kompresi 3 adalah LZSS (Lempel–Ziv–Storer–Szymanski) +BWT.

Tabel 5-10 Hasil Uji Coba 3

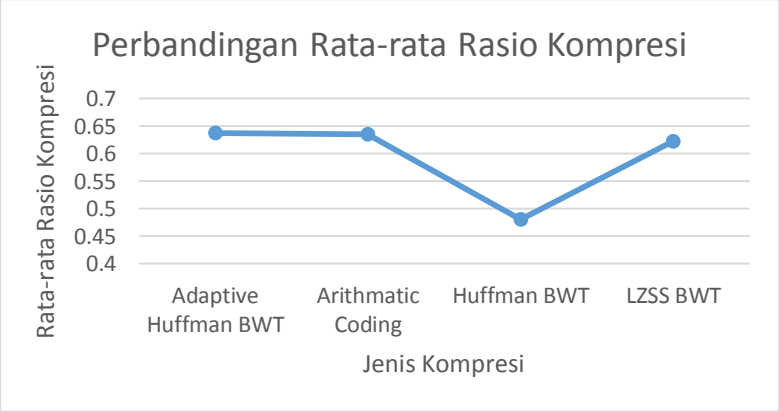
No	Berkas Pesan	Ukuran Sebelum (Byte)	Ukuran Sesudah (byte)	Rasio Kompresi
1	bib.txt	111.261	72.511	0,651
2	geo.txt	102.400	72.841	0,711
3	obj1.txt	21.504	16.383	0,761
4	paper1.txt	53.161	33.435	0,628
5	paper2.txt	82.199	47.611	0,579
6	paper3.txt	46.526	27.413	0,589
7	paper4.txt	13.286	7.979	0,600
8	paper5.txt	11.954	7.561	0,632
9	paper6.txt	38.105	24.047	0,631
10	progc.txt	39.611	26.062	0,657
11	progl.txt	71.646	42.783	0,597
12	progp.txt	49.379	30.354	0,614

Tabel 5-11 Perbandingan rasio kompresi uji coba 3

No	Berkas Pesan	A. Huff BWT	Komp. 1	Komp. 2	Komp. 3
1	bib.txt	0,651	0,654	0,457	0,627
2	geo.txt	0,711	0,707	0,725	0,788
3	obj1.txt	0,761	0,746	0,596	0,661
4	paper1.txt	0,628	0,623	0,452	0,622
5	paper2.txt	0,579	0,578	0,460	0,642
6	paper3.txt	0,589	0,589	0,482	0,667

No	Berkas Pesan	A. Huff BWT	Komp. 1	Komp. 2	Komp. 3
7	paper4.txt	0,600	0,603	0,508	0,672
8	paper5.txt	0,632	0,633	0,507	0,657
9	paper6.txt	0,631	0,626	0,454	0,619
10	progc.txt	0,657	0,655	0,438	0,591
11	progl.txt	0,597	0,595	0,335	0,456
12	progp.txt	0,614	0,612	0,345	0,461
Rata-rata rasio		0,637	0,635	0,480	0,622

Berdasarkan data pada tabel di atas, dapat dilihat bahwa performa dari teknik kompresi memiliki rata-rata rasio kompresi 0,637. Dimana rata-rata tersebut lebih buruk dibandingkan rata-rata lain yaitu 0,635; 0,480; 0,622. Namun pada beberapa jenis berkas seperti 'paper4' terlihat bahwa rasio kompresi pada metode Adaptive Huffman lebih baik dibandingkan metode *Arithmatic Coding* dan LZSS dengan rasio sebesar 0,600 berbanding 0,603 (*Arithmatic Coding*) dan 0,672 (LZSS). Grafik perbandingan rata-rata rasio kompresi dapat dilihat pada Gambar 5.16 di bawah ini.



Gambar 5.16 Grafik perbandingan rata-rata rasio kompresi

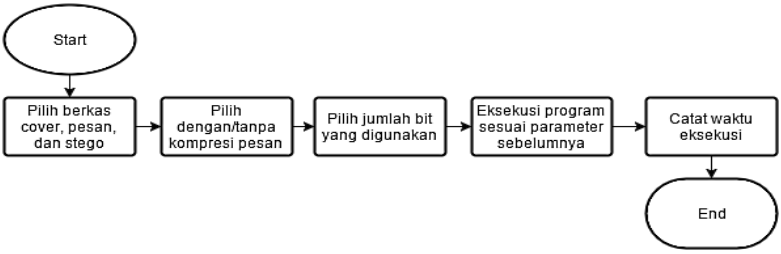
5.3.7 Skenario Uji Coba 4

Skenario uji coba 4 adalah melakukan perhitungan perhitungan waktu yang digunakan pada saat proses *insertion* berlangsung, baik menggunakan kompresi maupun tidak menggunakan kompresi. Dataset yang digunakan adalah 1 jenis teks dan 1 jenis musik dengan 3 resolusi yang berbeda. Spesifikasi dataset yang akan digunakan pada proses uji coba ini dapat dilihat pada Tabel 5-12.

Tabel 5-12 Spesifikasi dataset

No	Nama	Jenis	Durasi (s)	Resolusi (bit)	Ukuran (byte)
1	Bib.txt	Teks	-	-	111.261
2	Other8.wav	Audio	54	8	4.836.140
3	Other16.wav	Audio	54	16	9.672.236
4	Other24.wav	Audio	54	24	14.508.332

Penggunaan beragam jenis resolusi berkas dan beragam jumlah bit dilakukan untuk menganalisis waktu yang digunakan dalam memproses berkas. Selain itu, perbandingan waktu juga dilakukan untuk mengetahui performa kompresi yang digunakan.



Gambar 5.17 Skenario uji coba 4

Sesuai dengan Gambar 5.17 di atas, alur jalannya pengujian dimulai dengan memilih berkas *cover*, berkas pesan dan berkas *stego* yang akan digunakan untuk eksekusi. Kemudian

proses dilanjutkan dengan memilih pilihan untuk menggunakan kompresi. Selanjutnya tentukan panjang *bit* yang akan digunakan. Lakukan proses insertion dan catat nilai waktu yang digunakan selama pengujian pengujian berlangsung

5.3.8 Evaluasi Uji Coba 4

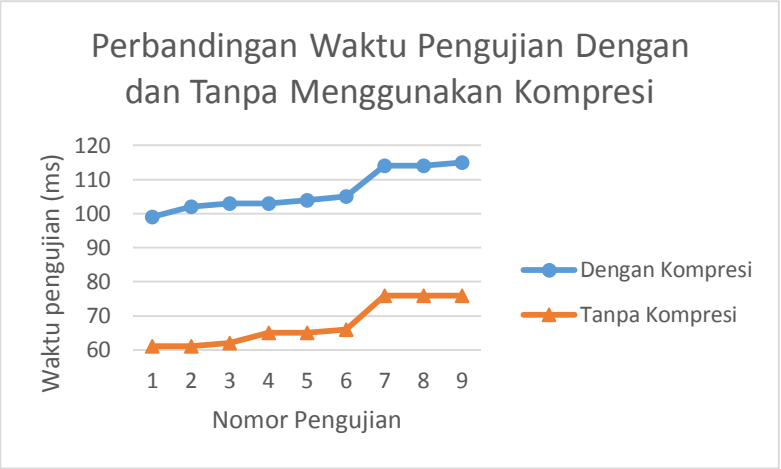
Setelah dilakukan uji coba sesuai skenario uji coba 4, didapatkan waktu pengujian yang berbeda-beda untuk setiap perbedaan aspek pengujian. Hasil uji coba proses dengan dan tanpa menggunakan kompresi data dapat dilihat pada Tabel 5.13.

Tabel 5-13 Perbandingan waktu pengujian performa

No	Batas Bit	Resolusi (Bit)	Frame Terpakai		Waktu Pengujian (ms)	
			Dengan Komp.	Tanpa Komp.	Dengan Komp.	Tanpa Komp.
1	1	8	580125	890125	99,00	61,00
2	2	8	290081	445081	102,00	61,00
3	3	8	193400	296733	103,00	62,00
4	1	16	580125	890125	103,00	65,00
5	2	16	290081	445081	104,00	65,00
6	3	16	193400	296733	105,00	66,00
7	1	24	580125	890125	114,00	76,00
8	2	24	290081	445081	114,00	76,00
9	3	24	193400	296733	115,00	76,00
Rata-Rata Waktu Pengujian					106,56	67,56

Berdasarkan tabel tersebut, dapat disimpulkan bahwa waktu pengujian berubah secara signifikan pada saat resolusi yang digunakan berbeda-beda dan pada saat proses dilakukan dengan atau tanpa menggunakan kompresi. Namun, jumlah *bit* yang digunakan tidak mempengaruhi hasil pengujian. Hal ini terlihat

pada waktu dalam 1 jenis resolusi dengan jumlah *bit* yang berbeda yang memiliki nilai yang saling berdekatan. Grafik perbandingan waktu pengujian dapat dilihat pada Gambar 5.18.



Gambar 5.18 Grafik perbandingan waktu pengujian antara dengan dan tanpa menggunakan kompresi

5.3.9 Skenario Uji Coba 5

Skenario uji coba 5 adalah melakukan pengujian subjektif terhadap kualitas berkas *stego* berdasarkan rata-rata nilai opini responden. Responden terdiri dari 2 orang pendengar yang peka terhadap nada dan 2 orang pendengar awam.

Pengujian dilakukan dengan cara meminta responden untuk mendengarkan 4 buah berkas suara dan mengisi kuisioner yang disediakan. Deskripsi berkas suara dan kuisioner dapat dilihat pada Tabel 5-15 dan Tabel 5-16. Tiap kolom memiliki bobot penilaian dengan ketentuan yang dapat dilihat pada Tabel 5-17.

Tabel 5-14 Deskripsi berkas suara

No	Nama	Deskripsi
1	1.wav	Berkas asli tanpa modifikasi

No	Nama	Deskripsi
2	2.wav	Berkas stego 8 bit dengan PSNR tertinggi
3	3.wav	Berkas stego 16 bit dengan PSNR tertinggi
4	4.wav	Berkas stego 24 bit dengan PSNR tertinggi

Tabel 5-15 Pertanyaan pada kuisioner

No	Pertanyaan
1.	Apakah perbedaan pada suara 1 dan suara 2 mengganggu anda?
2.	Apakah perbedaan pada suara 1 dan suara 3 mengganggu anda?
3.	Apakah perbedaan pada suara 1 dan suara 4 mengganggu anda?

Tabel 5-16 Bobot pertanyaan kuisioner

No	Jawaban	Bobot
1.	Sangat Mengganggu	1
2.	Mengganggu	2
3.	Tidak Mengganggu	3
4.	Tidak Terdengar	4

5.3.10 Evaluasi Uji Coba 5

Berdasarkan pengujian yang dilakukan sesuai dengan skenario uji coba 5. Didapatkan hasil bahwa kualitas berkas *stego* 3 dan 4 lebih baik dibandingkan kualitas berkas *stego* 2. Hal ini disebabkan oleh terdengarnya derau yang sangat mengganggu pada berkas *stego* 2 dimana pada berkas *stego* 3 dan 4 derau umumnya tidak menganggu atau tidak terdengar sama sekali. Jawaban uji coba para responden dapat dilihat pada lampiran B.

Tabel 5-17 Nilai opini rata-rata dari setiap pertanyaan

No	Pertanyaan	Nilai Opini Rata-Rata
1.	Apakah perbedaan pada suara 1 dan suara 2 mengganggu anda?	1,5

No	Pertanyaan	Nilai Opini Rata-Rata
2.	Apakah perbedaan pada suara 1 dan suara 3 mengganggu anda?	3,5
3.	Apakah perbedaan pada suara 1 dan suara 4 mengganggu anda?	3,5

Dari hasil pada Tabel 5-17 dapat disimpulkan bahwa berkas dengan resolusi 16 *bit* dan 24 *bit* cocok digunakan sebagai *cover* dalam melakukan steganografi dengan metode LSB.

LAMPIRAN A
KODE SUMBER

Pada lampiran berikut dituliskan potongan kode sumber yang digunakan. Kode sumber menggunakan Bahasa Pemrograman C++. Berikut kode sumber yang disertakan:

- Kode Sumber 1 : `Embed Handler`, yakni kode sumber prosedur utama proses penyisipan berkas yang akan memanggil prosedur *preprocessing*, kompresi, dan prosedur *insertion.*
- Kode Sumber 2: `Transform BWT`, yaitu kode sumber prosedur yang bertugas untuk melakukan transformasi pesan.
- Kode Sumber 3: `Encode Symbol Adaptive Huffman`, bertugas melakukan encoding pada berkas hasil transformasi
- Kode Sumber 4: `Write Message`, yaitu kode sumber yang melakukan proses penyisipan pada berkas *cover*.
- Kode Sumber 5: `Extract Handler`, yakni kode sumber prosedur utama proses pengambilan pesan yang akan memanggil prosedur *extraction*, dekompresi, dan prosedur reversi BWT.
- Kode Sumber 6: `Extract Message`, yaitu kode sumber yang melakukan proses pengambilan berkas pesan pada berkas *cover*.
- Kode Sumber 7: `Decode Symbol Adaptive Huffman`, bertugas melakukan decoding pada berkas hasil pengambilan
- Kode Sumber 8: `Reversi BWT`, yaitu kode sumber prosedur yang bertugas untuk melakukan reversi transformasi BWT dan menyimpan hasilnya sebagai pesan.

Kode Sumber 1 Embed Handler

```
1  void embedhandler::embedMessage(EmbedJob &job)
2  {
3      std::string strFilename =
4  job.messageFile.toStdString();
5      const char *msgFilename =strFilename.c_str();
6      QString embedMsgBitString;
7      FILE * msg = fopen(msgFilename,"rb");
8      msg = util::compressMessage(&msg,job.compType);
9      fseek(msg,0,SEEK_END);
10     embedMsgBuffLen=ftell(msg);
11     rewind(msg);
12     embedMsgBuff = (char*) malloc
   (sizeof(char)*embedMsgBuffELen);
13     fread(embedMsgBuff,1, embedMsgBuffLen,msg);
14     embedMsgBitString =
   createExtractInfo((int)embedMsgBuffLen,
   job.compType,job.bitnumber);
15     std::string tempMsgBuff =
   util::charArrToBin(embedMsgBuff, embedMsgBuffLen);
16     embedMsgBitString.append(Qstring::
   fromStdString(tempMsgBuff));
17     embedSoundFileInfo.format = 0;
18     std::string tempFilename =
   job.sourceFile.toStdString();
19     embedOutSoundFileInfo=embedSoundFileInfo

20 if(!checkSize(job,embedSoundFileInfo,(long)tempMsgBuff
   .length()))
21     {
22         result.status = -1;
23         fclose(msg);
24         sf_close(embedSf);
25         reset();
26         return;
27     }
28     tempFilename = job.outputFile.toStdString();
29     int embedSoundSubFormat =
   embedSoundFileInfo.format & SF_FORMAT_SUBMASK;
30     switch (embedSoundSubFormat) {
31     case SF_FORMAT_PCM_16:
32         writeOtherInt(job,embedMsgBitString,16);
33         break;
34     case SF_FORMAT_PCM_U8:
35         unsignedOperation=1;
36         writeOtherInt(job,embedMsgBitString,8);
37         unsignedOperation=0;
38         break;
39     case SF_FORMAT_PCM_24:
```

```
40              writeOtherInt(job,embedMsgBitString,24);
41              break;
42          }
43          fclose(msg);
44          sf_close(embedSf);
45          sf_close(embedOutSf);
46          reset();
47          return;
48      }
```

Kode Sumber 2 Encode Symbol Adaptive Huffman

```
 1  void EncodeSymbol( TREE *tree, unsigned int c,
    BIT_FILE *output )
 2  {
 3      unsigned long code;
 4      unsigned long current_bit;
        int code_size;
 5      int current_node;

 6      code = 0;
 7      current_bit = 1;
 8      code_size = 0;
 9      current_node = tree->leaf[ c ];
10      if ( current_node == -1 )
11          current_node = tree->leaf[ ESCAPE ];
12      while ( current_node != ROOT_NODE ) {
13          if ( ( current_node & 1 ) == 0 )
14              code |= current_bit;
15          current_bit <<= 1;
16          code_size++;
17          current_node = tree-
    >nodes[ current_node ].parent;
18      };
19      OutputBits( output, code, code_size );
20      if ( tree->leaf[ c ] == -1 ) {
21          OutputBits( output, (unsigned long) c, 8 );
            add_new_node( tree, c );
22      }
23  }
```

Kode Sumber 3 Tranformasi BWT

```
1  void BWT::preprocessBWT(FILE* messageFile, FILE*
   outFile)
2  {
3      int originalCharPosition;
4      FILE * file = messageFile;
5      FILE * output = outFile;
6      int i;
7      block = (char *)malloc(BLOCK_SIZE * sizeof(char));
8      baris = (int *) malloc(BLOCK_SIZE * sizeof(int));
9      last = (char *) malloc(BLOCK_SIZE * sizeof(char));
10     blockLength = fread((char *)
   block,1,BLOCK_SIZE,file);
11     while(blockLength>0)
12     {
13         memset(last,0,BLOCK_SIZE*sizeof(char));
14         memset(baris,0,BLOCK_SIZE*sizeof(int));
15         for(i=0;i<blockLength;i++)
16         {
17             baris[i]=i;
18         }
19         qsort(&baris[0],blockLength,sizeof(int),
   BoundedCompare);
20         for(i=0;i<blockLength;i++)
21         {
22             if(baris[i]==1)
23             {
24                 originalCharPosition = i;
25             }
26             last[i]=getLastColumnChar(baris[i],(int)
27 blockLength);
           }
28
           fwrite(&originalCharPosition,sizeof(int),1,
   output);
29         fwrite(last,sizeof(char),blockLength,output);
           blockLength=fread((char *)
30 block,1,BLOCK_SIZE,file);
       }
31     free(last);
32     free(block);
33     free(baris);
34     return;
35 }
36
```

Kode Sumber 4 Write Message

```
1  void embedhandler::writeOtherInt(EmbedJob job, QString
   bitstring, int length)
2  {
3      EmbedResult embedResult;
4      int adaptiveColumnPosition=0;
5      int * embedAudioBuff;
6      embedAudioBuff = (int*)malloc(4096*sizeof(int));
7      int read;
8      int written;
9      read = sf_readf_int(embedSf, embedAudioBuff,
   4096/embedSoundFileInfo.channels);
10     embedBitCounter=0;
11     while(read!=0){
12         embedRowCounter=0;
13         while(embedRowCounter < read *
14 embedSoundFileInfo.channels){
15             embedColumnCounter=0;
16             if(embedBitCounter<37){
17                 embedBitNumber=1;
18             }
19             else{
20                 embedBitNumber=job.bitnumber;
21             }
22             while(embedColumnCounter<embedBitNumber){
                   adaptiveColumnPosition = (sizeof(int)
   * CHAR_BIT) - length + embedColumnCounter;
23                 if(embedBitCounter <
   bitstring.length()){
24                     if(bitstring[embedBitCounter] ==
   '0'){
25                         if((embedAudioBuff
   [embedRowCounter] & (1 << adaptiveColumnPosition))!=
   0) {
26                             embedAudioBuff
   [embedRowCounter] ^= 1 << adaptiveColumnPosition;}
27                     }
29                     else{
29                         if((embedAudioBuff
   [embedRowCounter] & (1 << adaptiveColumnPosition))==
   0){
30                             embedAudioBuff
   [embedRowCounter] ^= 1 << adaptiveColumnPosition;
31                         }
32                     }
33                     embedBitCounter++;
34                 }
35                 else{
36                     break;
```

```
37                  }
38                  embedColumnCounter++;
39              }
40              embedRowCounter++;
41          }
42          written=sf_writef_int(embedOutSf,
    embedAudioBuff,read);
43          read = sf_readf_int (embedSf,embedAudioBuff,
    4096/embedSoundFileInfo.channels);
45      }
46      free(embedAudioBuff);
47  }
```

Kode Sumber 5 Extract Handler

```
1  void extracthandler::extractMessage(ExtractJob &job)
2  {
3      QString tempRawMsgFileName = util::tempDir.path();
4      tempRawMsgFileName.append("/tempRawFile.txt");
5      std::string tempFilename =
   tempRawMsgFileName.toStdString();
6      extractRawMsg = fopen(tempFilename.c_str(),"wb");
7      tempFilename = job.inputEmFile.toStdString();
8      extractSf = sf_open(tempFilename.c_str() ,
   SFM_READ,&extractSoundFileInfo);
9      int extractSoundSubformat =
   extractSoundFileInfo.format & SF_FORMAT_SUBMASK;;
10     switch (extractSoundSubformat) {
11         case SF_FORMAT_PCM_16:
12             readShort(job);
13             break;
14         case SF_FORMAT_PCM_U8:
15             readOtherInt(job,8);
16             break;
17         case SF_FORMAT_PCM_24:
18             readOtherInt(job,24);
19             break;
20         default:
21             break;
22         }
23     fclose(extractRawMsg);
24     sf_close(extractSf);
25     FILE * extractOutMsg = util::decompressMessage
   (tempRawMsgFileName,job.compType);
27     rewind(extractOutMsg);
28     tempFilename=job.outputExFile.toStdString();
29     FILE * result = fopen(tempFilename.c_str(),"wb");
30     util::CopyFile(extractOutMsg,result);
31     reset();
32     return;
33 }
```

Kode Sumber 6 Extract Message

```
1  void readOtherInt(ExtractJob &job,int length)
2  {
3      int read;
4      int written;
5      int * extractAudioBuff;
6      extractAudioBuff = (int*)
   malloc(sizeof(int)*4096);
7      read = sf_readf_int(extractSf, extractAudioBuff,
   4096/extractSoundFileInfo.channels);
8      char * charFromBin;
9      extractBitCounter=0;
10     QString bitstring;
11     std::string extractedMessage;
12     job.bitNumber=1;
13     job.msgSize=INT_MAX;
14     int extractedInfo=0;
15     int done=0;
16     int adaptiveColumnPositon=0;
17     while((read!=0) && (done==0))
       {
18         extractRowCounter=0;
19         while((extractRowCounter < read *
   extractSoundFileInfo.channels) && (done==0) )
20         {
21             if(extractBitCounter==37 &&
   extractedInfo==0)
22             {
23                 readExtractInfo(bitstring,job);
24                 extractedInfo=1;
25             }
26             extractColumnCounter=0;
27             while(extractColumnCounter <
   job.bitNumber)
28             {
29                 adaptiveColumnPositon = (sizeof(int) *
   CHAR_BIT) - length + extractColumnCounter;
30                 if(extractBitCounter<job.msgSize)
31                 {
32                     if((extractAudioBuff
   [extractRowCounter] & ( 1 << adaptiveColumnPositon))!=
   0)
33                     {
34                         bitstring.append("1");
35                     }
36                     else
37                     {
38                         bitstring.append("0");
39                     }
```

```
40                      extractBitCounter++;
41                  }
42                  else
43                  {
44                      done=1;
45                      break;
46                  }
47                  extractColumnCounter++;
48              }
49              extractRowCounter++;
50          }
51          read = sf_readf_int (extractSf ,
    extractAudioBuff,4096/extractSoundFileInfo.channels);
52      }
53      extractedMessage = bitstring.toStdString();
54      charFromBin = util::binToCharArr
    (extractedMessage);
55      written = (int)fwrite(charFromBin , 1 ,
    job.msgSize/CHAR_BIT,extractRawMsg);
56      free(extractAudioBuff);
57      return;
58  }
```

Kode Sumber 7 Decode Symbol Adaptive Huffman

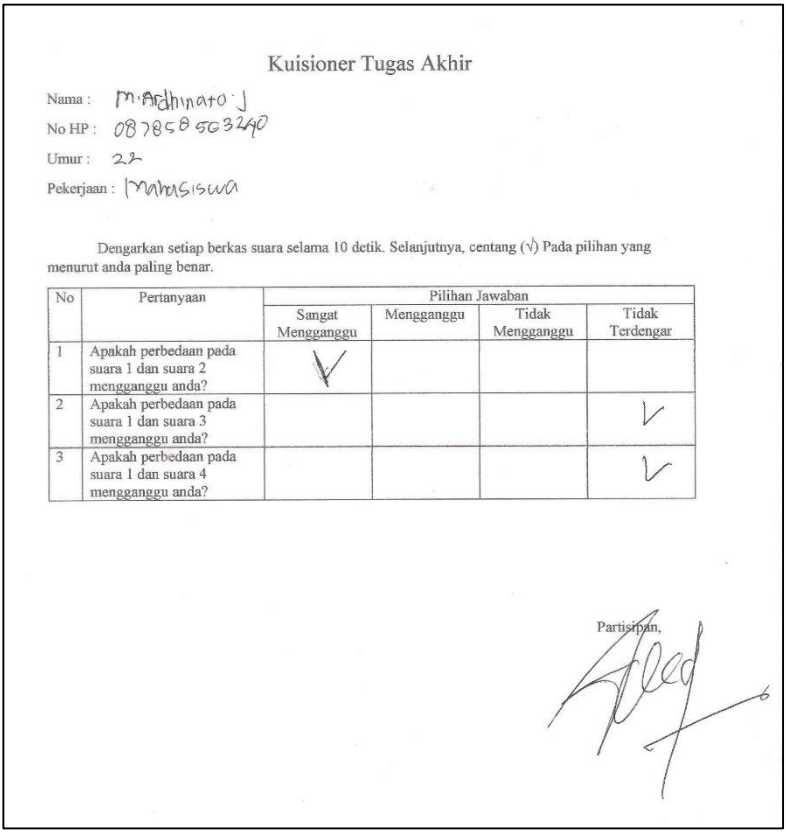
```
1   int DecodeSymbol( TREE *tree, BIT_FILE *input )
2   {
3       int current_node;
4       int c;

5       current_node = ROOT_NODE;
6       while ( !tree-
    >nodes[ current_node ].child_is_leaf ) {
7           current_node = tree-
    >nodes[ current_node ].child;
8           current_node += InputBit( input );
9       }
10      c = tree->nodes[ current_node ].child;
11      if ( c == ESCAPE ) {
12          c = (int) InputBits( input, 8 );
13          add_new_node( tree, c );
14      }
15      return( c );
16  }
```

Kode Sumber 8 Reversi BWT

```
1  void BWT::reverseBWT(FILE* inFile, FILE* outFile)
2  {
3      int originalCharPosition;
4      block = (char *)malloc(BLOCK_SIZE * sizeof(char));
5      sorted = (char *)malloc(BLOCK_SIZE *
   sizeof(char));
6      result = (char *)malloc(BLOCK_SIZE *
   sizeof(char));
7      transform = (int *)malloc(BLOCK_SIZE *
   sizeof(int));
8      int i,j;
9      while(fread(&originalCharPosition,sizeof(int),1,
   inFile)!=0){
10         blockLength = fread(block,sizeof(char),
   BLOCK_SIZE,inFile);
11         memcpy(sorted,block,sizeof(char)*blockLength);
12         qsort(sorted,blockLength,sizeof(char),
   sortArray);
13         memset(lastpos,0, sizeof(int)*UCHAR_MAX);
14         for(i=0;i<blockLength;i++){
15             j=lastpos[sorted[i]];
16             while(1){
17                 if(j>=blockLength){
18                     break;
19                 }
20                 if(block[j]==sorted[i]){
21                     transform[i]=j;
22                     lastpos[sorted[i]]=j+1;
23                     break;
24                 }
25                 j++;
26             }
27         }
28         int index=originalCharPosition;
29         for(i=0;i<blockLength;i++){
30             result[i]=block[index];
31             index=transform[index];
32         }
33         fwrite(result,sizeof(char),blockLength,
   outFile);
34     }
35     free(result);
36     free(block);
37     free(sorted);
38     free(transform);
39     return;
40 }
```

LAMPIRAN B
JAWABAN KUISIONER

Pada lampiran ini diberikan jawaban kuisioner dari setiap responden dalam bentuk gambar. Terdapat 3 soal dengan jawaban yang berbeda dari 4 responden yang telah mengikuti proses uji coba subjektif.



Kuisioner Tugas Akhir

Nama : M. Ardhinata J

No HP : 087858563240

Umur : 22

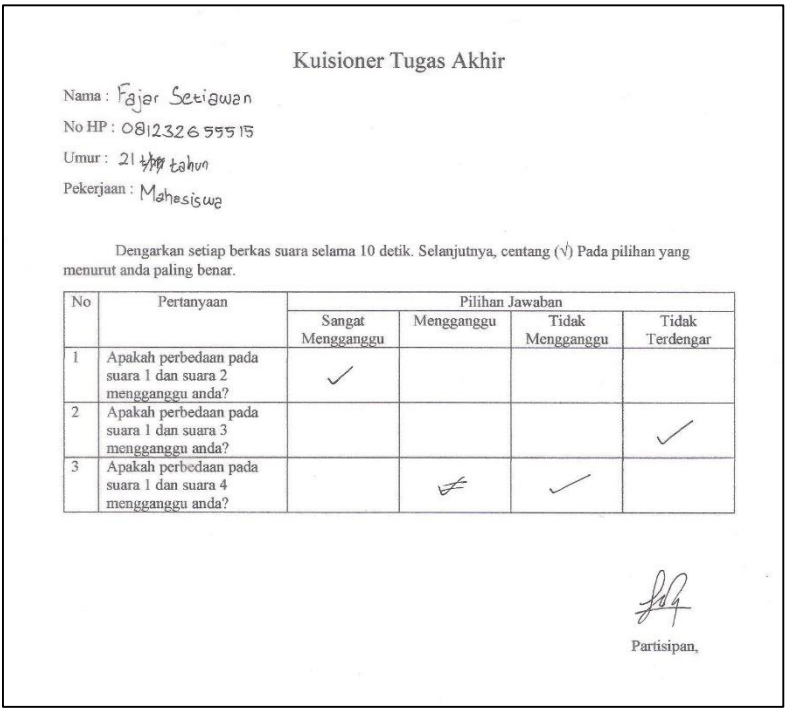
Pekerjaan : Mahasiswa

Dengarkan setiap berkas suara selama 10 detik. Selanjutnya, centang (√) Pada pilihan yang menurut anda paling benar.

No	Pertanyaan	Pilihan Jawaban			
		Sangat Mengganggu	Mengganggu	Tidak Mengganggu	Tidak Terdengar
1	Apakah perbedaan pada suara 1 dan suara 2 mengganggu anda?	√			
2	Apakah perbedaan pada suara 1 dan suara 3 mengganggu anda?				√
3	Apakah perbedaan pada suara 1 dan suara 4 mengganggu anda?				√

Partisipan,

Gambar B.1 Jawaban Kuisioner M. Ardhinata Juari



Kuisioner Tugas Akhir

Nama : Fajar Setiawan

No HP : 0812326555 15

Umur : 21 tahun

Pekerjaan : Mahasiswa

Dengarkan setiap berkas suara selama 10 detik. Selanjutnya, centang (√) Pada pilihan yang menurut anda paling benar.

No	Pertanyaan	Pilihan Jawaban			
		Sangat Mengganggu	Mengganggu	Tidak Mengganggu	Tidak Terdengar
1	Apakah perbedaan pada suara 1 dan suara 2 mengganggu anda?	✓			
2	Apakah perbedaan pada suara 1 dan suara 3 mengganggu anda?				✓
3	Apakah perbedaan pada suara 1 dan suara 4 mengganggu anda?		✓	✓	

Partisipan,

Gambar B.2 Jawaban Kuisioner Fajar Setiawan

Kuisioner Tugas Akhir

Nama : M. Y. Rohmat

No HP : 082112564593

Umur : 21

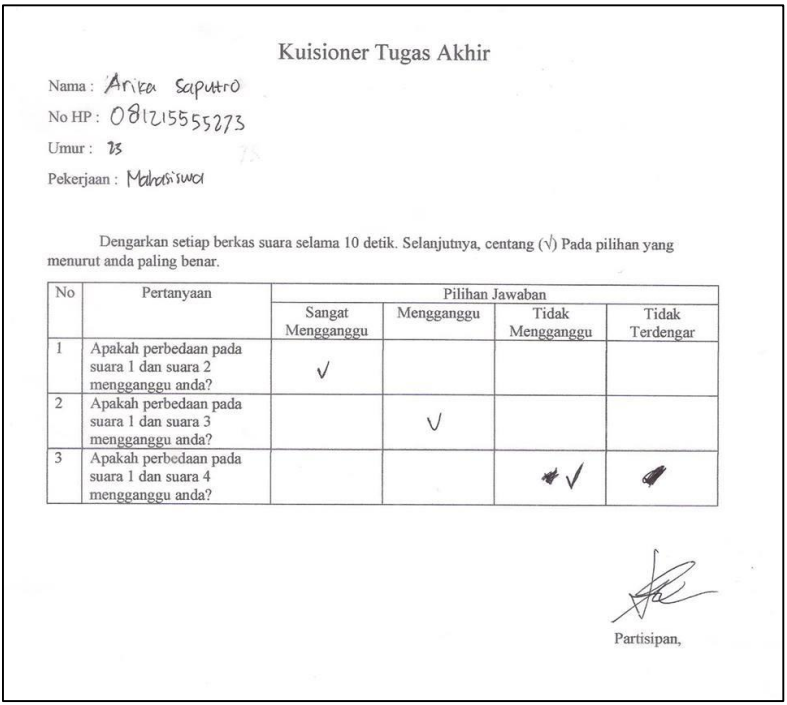
Pekerjaan : Mahasiswa

Dengarkan setiap berkas suara selama 10 detik. Selanjutnya, centang (√) Pada pilihan yang menurut anda paling benar.

No	Pertanyaan	Pilihan Jawaban			
		Sangat Mengganggu	Mengganggu	Tidak Mengganggu	Tidak Terdengar
1	Apakah perbedaan pada suara 1 dan suara 2 mengganggu anda?		√		
2	Apakah perbedaan pada suara 1 dan suara 3 mengganggu anda?				√
3	Apakah perbedaan pada suara 1 dan suara 4 mengganggu anda?				√

Partisipan,

Gambar B.3 Jawaban Kuisioner M. Yarjuna Rohmat



Kuisioner Tugas Akhir

Nama : Arika Saputro

No HP : 081215555273

Umur : 23

Pekerjaan : Mahasiswa

Dengarkan setiap berkas suara selama 10 detik. Selanjutnya, centang (√) Pada pilihan yang menurut anda paling benar.

No	Pertanyaan	Pilihan Jawaban			
		Sangat Mengganggu	Mengganggu	Tidak Mengganggu	Tidak Terdengar
1	Apakah perbedaan pada suara 1 dan suara 2 mengganggu anda?	√			
2	Apakah perbedaan pada suara 1 dan suara 3 mengganggu anda?		√		
3	Apakah perbedaan pada suara 1 dan suara 4 mengganggu anda?			√	

Partisipan,

Gambar B.4 Jawaban Kuisioner Arika Saputro

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan pada bab sebelumnya, yaitu Bab Uji Coba dan Evaluasi. Bab ini juga digunakan sebagai jawaban dari rumusan masalah yang dikemukakan pada Bab Pendahuluan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1 Kesimpulan

Dari proses pengerjaan tahap perancangan, implementasi, dan pengujian yang dilakukan terhadap program dapat diambil beberapa kesimpulan sebagai berikut:

1. Pesan dapat disisipkan dalam berkas audio dengan cara mengubah nilai LSB. Ekstraksi dapat dilakukan dengan cara membaca kembali nilai LSB pada berkas hasil proses penyisipan.
2. Ukuran pesan yang disimpan dalam berkas audio dapat dikurangi menggunakan algoritma kompresi Adaptive Huffman dengan *preprocessing* BWT sehingga mengurangi jumlah LSB yang diubah dan meningkatkan nilai PSNR dan SNR pada berkas hasil penyisipan.
3. Performa kompresi Adaptive Huffman dikombinasikan dengan *preprocessing* BWT memiliki rata-rata rasio kompresi 0,637 yang lebih buruk dibandingkan dengan kompresi lain. Namun, pada beberapa berkas performa kompresi lebih baik dibandingkan kompresi lainnya.
4. Performa program yang dikembangkan lebih baik pada saat tidak menggunakan kompresi dengan rata-rata waktu yang diperlukan 67,56 ms dibandingkan dengan menggunakan kompresi yaitu 106,56 ms.

5. Berkas *stego* dengan resolusi 16 *bit* dan 24 *bit* memiliki kualitas lebih baik dibandingkan resolusi 8 *bit*.

6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

1. Sebaiknya digunakan beragam jenis dataset sebagai berkas pesan seperti gambar, suara, atau video.
2. Menggunakan teknik kompresi dengan hasil yang lebih baik dibandingkan dengan Adaptive Huffman.
3. Menggunakan teknik *preprocessing* dengan nilai keluaran yang dapat meningkatkan kinerja teknik kompresi yang digunakan.
4. Menggunakan beragam jenis berkas suara dengan properti yang beragam sebagai berkas *cover*.
5. Meningkatkan kualitas hasil steganografi LSB tidak hanya dengan memanipulasi berkas pesan. Namun dengan menambahkan fitur lain pada saat proses penyisipan berlangsung. Sebagai contoh mendeteksi apakah *frame* yang akan disisipkan berada diantara *frame* dengan nilai yang cukup tinggi untuk menyembunyikan perubahan yang dilakukan.

DAFTAR PUSTAKA

[1] Y. Venkataramani dan M. B. Begum, "LSB Based Audio Steganography Based On Text Compression," *Procedia Engineering*, no. 30, pp. 703-710, 2011.

[2] N. Cvejic dan T. Seppänen, "Increasing the capacity of LSB-based audio steganography," *2002 IEEE Workshop on Multimedia Signal Processing*, pp. 336-338, 2002.

[3] I. Mengyi Pu, Fundamental Data Compression, Burlington: Elsevier, 2006.

[4] D. Salomon dan G. Motta, Handbook of Data Compression, 5th penyunt., London: Springer, 2010.

[5] A. Binny dan M. Koilakuntla, "Hiding Secret Information Using LSB Based Audio Steganography," dalam *International Conference on Soft Computing & Machine Intelligence*, 2014.

[6] K. Gopalan, "AUDIO STEGANOGRAPHY USING BIT MODIFICATION," dalam *International Conference on Acoustics, Speech, & Signal Processing*, Hong Kong, 2003.

[7] W. Bender, D. Gruhl, N. Morimoto dan A. Lu, "Techniques for data hiding," *IBM SYSTEMS JOURNAL*, vol. 35, no. 3&4, pp. 313-336, 1996.

[8] U. Yavanoglu, B. Ozcakmak dan O. Milletsever, "A New Intelligent Steganalysis Method for Waveform Audio Files," *11th International Conference on Machine Learning and Applications*, pp. 233-239, 2012.

[9] A. Kanhe, G. Aghila, C. Y. S. Kiran, C. H. Ramesh, G. Jadav dan M. G. Raj, "Robust Audio Steganography based on Advanced Encryption Standards in Temporal Domain," dalam *International Conference on Advances in Computing, Communications and Informatics*, 2015.

[10] M. Powell, “The Canterbury Corpus,” University of Canterbury, 20 December 2000. [Online]. Available: http://corpus.canterbury.ac.nz/descriptions/#calgary. [Diakses 4 June 2016].

[11] M. Nelson, The Data Compression Book, Cambridge: IDG Books Worldwide, Inc..

BIODATA PENULIS

Rahmat Irfan, lahir pada tanggal 21 Januari 1996 di Muara Enim. Penulis telah menempuh pendidikan formal pada SD Kartika Jaya Palembang (2001-2003), SDN 016 Rengat (2003-2007), SMPN 1 Rengat (2007-2008), SMP Santa Maria Pekanbaru (2008-2010), SMAN 8 Pekanbaru (2010-2012), dan S1 di Jurusan Teknik Informatika, Institut Teknologi Sepuluh Nopember Surabaya, dengan rumpun mata kuliah Komputasi Berbasi Jaringan. Selama masa perkuliahan, penulis pernah menjadi panitia Acara di ITS EXPO 2013 dan pernah menjadi panitia dalam acara LKMM-Pra TD VI FTIF 2013. Pada jenjang SMA, penulis aktif mengikuti organisasi Palang Merah Remaja serta menjadi juara 1 dalam Olimpiade TIK Tingkat Kota Pekanbaru.

Kritik dan saran sangat diharapkan guna peningkatan kualitas dan penulisan selanjutnya. Untuk itu, silahkan kirim kritik dan saran ke : **irfanrahmat007@gmail.com**