

**STUDI KOMPARASI ALGORITMA SEQUITUR DAN ALGORITMA
MOVE-TO-FRONT CODING DALAM KOMPRESI FILE TEKS**

SKRIPSI

DONISIUS MARTIN SIRAIT

171401129



**PROGRAM STUDI S-1 ILMU KOMPUTER
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS SUMATERA UTARA
MEDAN
2024**

**STUDI KOMPARASI ALGORITMA SEQUITUR DAN ALGORITMA
MOVE-TO-FRONT CODING DALAM KOMPRESI FILE TEKS**

SKRIPSI

DONISIUS MARTIN SIRAIT

171401129



**PROGRAM STUDI S-1 ILMU KOMPUTER
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS SUMATERA UTARA
MEDAN
2024**

PERSETUJUAN

Judul : STUDI KOMPARASI ALGORITMA SEQUITUR
 DAN ALGORITMA MOVE-TO-FRONT CODING
 DALAM KOMPRESI FILE TEKS

Kategori : SKRIPSI

Nama : DONISIUS MARTIN SIRAIT

Nomor Induk Mahasiswa : 171401129

Program Studi : SARJANA (S1) ILMU KOMPUTER

Fakultas : ILMU KOMPUTER DAN TEKNOLOGI
 INFORMASI UNIVERSITAS SUMATERA UTARA

Komisi Pembimbing :
 Tanggal Sidang : 24 Juni 2024

Dosen Pembimbing II

Amer Sharif, S.Si, M.Kom

NIP. 196910212021011001

Dosen Pembimbing I

Dr. Mohammad Andri Budiman S.T.,

M.Comp.Sc., M.E.M.

NIP. 197510082008011011

Diketahui/Disetujui oleh

Program studi S1 Ilmu Komputer



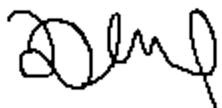
PERNYATAAN

STUDI KOMPARASI ALGORITMA SEQUITUR DAN ALGORITMA MOVE-TO-FRONT CODING DALAM KOMPRESI FILE TEKS

SKRIPSI

Saya mengakui bahwa skripsi ini adalah hasil karya saya sendiri, kecuali beberapa kutipan dan ringkasan yang masing-masing telah disebutkan sumbernya.

Medan, 24 Juni 2024



Donisius Martin Sirait

171401129

UCAPAN TERIMA KASIH

Segala puji syukur, hormat, serta kemuliaan bagi Tuhan, Allah Bapa Yang Maha Kuasa, atas berkat dan karunia-Nya lah penulis dapat menyelesaikan Tugas Akhir Mahasiswa S-1 Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara.

Dalam penelitian ini penulis tidak luput dari bantuan dan dukungan berbagai pihak yang terlibat dalam segala proses penggerjaan skripsi ini. Oleh karena itu, penulis secara khusus ingin mengucapkan rasa terima kasih yang sebesar-besarnya kepada:

1. Bapak Dr. Muryanto Amin, S.Sos., M.Si selaku Rektor Universitas Sumatera Utara.
2. Ibu Dr. Maya Silvi Lydia, M.Sc., selaku Dekan Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara.
3. Ibu Dr. Amalia, S.T., M.T selaku Ketua Program Studi S-1 Ilmu Komputer Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara.
4. Bapak Dr. M. Andri Budiman, S.T., M.Comp.Sc., M.E.M. selaku dosen pembimbing I yang senantiasa memberikan arahan, bimbingan serta dukungan kepada penulis dalam penggerjaan skripsi ini.
5. Bapak Amer Sharif, S.Si, M.Kom selaku dosen pembimbing II yang senantiasa memberikan nasihat serta bimbingan kepada penulis dalam menyelesaikan skripsi ini.
6. Seluruh Bapak dan Ibu dosen tenaga pengajar dan sivitas akademika Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara.
7. Bang David dan kakak Tercinta Debora Melani Sirait, yang tiada hentinya memberikan doa dan semangat kepada penulis dalam kehidupan ini.
8. Sahabat seperjuangan penulis dari NBSO yaitu Jeremy Michael Sinaga, Egi Wahyu Rasmamana Dakhi, dan Surya Andhika Ramadhani Ginting. Serta teman-teman Stambuk 2017 S-1 Ilmu Komputer USU terkhusus Kom C sebagai teman sekelas penulis.
9. Seluruh sahabat online penulis siapa dan dimana pun mereka berada, serta semua pihak yang telah mendukung dan membantu penulis yang tidak dapat disebutkan satu-persatu

ABSTRAK

Kemajuan teknologi telah berperan penting dalam mengubah cara manusia bertukar data dan informasi. Dari penggunaan media cetak, kini beralih ke media online, yang menuntut pengguna untuk memiliki akses cepat terhadap informasi. Namun, pergeseran ini juga menimbulkan tantangan baru terkait keterbatasan ruang penyimpanan. Salah satu solusi yang dapat diterapkan untuk mengatasi masalah ini adalah dengan menggunakan teknik kompresi data. Dengan kompresi data, informasi dapat disimpan dalam bentuk yang lebih efisien, memungkinkan pengguna untuk menghemat ruang penyimpanan tanpa kehilangan data yang penting. Algoritma MTF (Move-To-Front) Coding dan Algoritma Sequitur adalah dua teknik kompresi data yang memiliki pendekatan dan karakteristik masing-masing. MTF Coding bekerja dengan memanipulasi daftar simbol dan memindahkan simbol yang sering muncul ke posisi terdepan dalam daftar, yang mengurangi redundansi simbol yang sering muncul. Di sisi lain, Algoritma Sequitur mendeteksi pola berulang dalam data dan mengantikannya dengan aturan produksi, menciptakan grammar yang lebih kompak untuk representasi data. Dalam hal kinerja kompresi, MTF Coding cenderung lebih efektif untuk data dengan banyak simbol berulang dalam jangka pendek, sedangkan Algoritma Sequitur lebih efisien untuk data dengan pola berulang yang panjang dan kompleks. Pemilihan algoritma yang tepat bergantung pada jenis dan karakteristik data yang dikompresi, dengan tujuan untuk meningkatkan efisiensi kompresi dan mengoptimalkan waktu pemrosesan.

kata kunci: Desain dan Analisis Algoritma, Kompresi Data, *Sequitur*, *MTF Coding*, Dekompresi Data

COMPARATIVE STUDY OF SEQUITUR ALGORITHM AND MOVE-TO-FRONT

CODING ALGORITHM IN TEXT FILE COMPRESSION

ABSTRACT

The advancement of technology has played a crucial role in changing the way humans exchange data and information. From the use of print media, it has now shifted to online media, demanding users to have quick access to information. However, this shift also poses new challenges related to storage space limitations. One solution that can be applied to address this issue is by using data compression techniques. With data compression, information can be stored in a more efficient form, allowing users to save storage space without losing important data. Move-To-Front (MTF) Coding Algorithm and Sequitur Algorithm are two data compression techniques that have their own approaches and characteristics. MTF Coding works by manipulating a list of symbols and moving frequently occurring symbols to the front position in the list, reducing the redundancy of frequently occurring symbols. On the other hand, the Sequitur Algorithm detects repetitive patterns in the data and replaces them with production rules, creating a more compact grammar for data representation. In terms of compression performance, MTF Coding tends to be more effective for data with many repetitive symbols in the short term, while the Sequitur Algorithm is more efficient for data with long and complex repetitive patterns. The selection of the appropriate algorithm depends on the type and characteristics of the compressed data, with the goal of improving compression efficiency and optimizing processing time.

keyword: Design and Analysis of Algorithm, Data Compression, Sequitur, MTF Coding, Data Decompression

DAFTAR ISI

PERSETUJUAN	ii
PERNYATAAN	iii
UCAPAN TERIMA KASIH	iv
ABSTRAK	v
<i>COMPARATIVE STUDY OF SEQUITUR ALGORITHM AND MOVE-TO-FRONT CODING ALGORITHM IN TEXT FILE COMPRESSION</i>	vi
<i>ABSTRACT</i>	vi
DAFTAR ISI	vii
DAFTAR TABEL	ix
DAFTAR GAMBAR	x
DAFTAR LAMPIRAN	xi
BAB 1 PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Rumusan Masalah	3
1.3. Batasan Masalah	3
1.4. Tujuan Penelitian	4
1.5. Manfaat Penelitian	4
1.6. Penelitian Relevan	4
1.7. Metodologi Penelitian	6
1.8. Sistematika Penulisan	7
BAB 2 LANDASAN TEORI	8
2.1. Kompresi Data	8
2.2. Dekompresi Data	9
2.3. Pengukuran Kompresi	10
2.4. Teknik Kompresi	11
2.5. Algoritma <i>Sequitur</i>	13
2.6. Algoritma Move-to-Front (MTF) Coding	19
BAB 3 ANALISIS DAN PERANCANGAN SISTEM	26
3.1. Analisis Sistem	26
3.2. Pemodelan Sistem	29
3.3. Flow Chart	35
3.4. Tampilan Sistem	36

BAB 4 IMPLEMENTASI DAN PENGUJIAN SISTEM.....	40
4.1. Implementasi Sistem.....	40
4.2. Pengujian	43
4.3. Proses Pengujian Sistem.....	44
4.4. Hasil Pengujian.....	54
BAB 5 KESIMPULAN DAN SARAN	60
5.1. Kesimpulan.....	60
5.1. Saran	61
DAFTAR PUSTAKA.....	62

DAFTAR TABEL

Tabel 2.1. Contoh Proses Kompresi Sequitur.....	17
Tabel 2.2. Contoh Proses Perhitungan Kompresi Biasa	17
Tabel 2.3. Contoh Proses Perhitungan Kompresi Algoritma Sequitur	18
Tabel 2.4. Contoh Dekompresi Algoritma Sequitur	19
Tabel 2.5. Contoh Perhitungan Algoritma MTF CODING	23
Tabel 3.1. Keterangan Gambar Rancangan Halaman Menu Utama.....	37
Tabel 3.2. Keterangan Gambar Rancangan Halaman kompresi.....	38
Tabel 3.3. Keterangan Gambar Rancangan Halaman Dekompresi.....	50
Tabel 4.1. Daftar Bahan Uji.....	43
Tabel 4.2. Contoh proses kompresi sequitur	49
Tabel 4.3. Contoh Proses Perhitungan Sebelum Kompresi.....	49
Tabel 4.4. Contoh Proses Perhitungan Kompresi Algoritma Sequitur	50
Tabel 4.5. Tabel Dekompresi Sequitur	54
Tabel 4.6. Hasil Pengujian Kompresi.....	55
Tabel 4.7. Hasil Pengujian Dekompresi	55

DAFTAR GAMBAR

Gambar 2.1 Proses Kompresi dan Dekompresi Data Lossy.....	12
Gambar 2.2 Proses Kompresi dan Dekompresi Data Lossless.....	12
Gambar 3.1. Fish Bone Diagram.....	27
Gambar 3.2 Use Case Diagram	29
Gambar 3.3. Activity Diagram Proses Kompresi Algoritma Sequitur.....	30
Gambar 3.4. Activity Diagram Proses Kompresi Algoritma MTF Coding.....	31
Gambar 3.5. Activity Diagram Proses Dekompresi Algoritma Sequitur	32
Gambar 3.6. Activity Diagram Proses Dekompresi Algoritma MTF Coding.....	33
Gambar 3.7. Diagram Sequence.....	34
Gambar 4.1. Halaman Home	41
Gambar 4.2. Halaman Kompresi	42
Gambar 4.3. Halaman Dekompresi	42
Gambar 4.4. Tombol open untuk memilih file teks yang akan dikompresi.....	45
Gambar 4.5. Open File Dialog	45
Gambar 4.6. Tampilan dari isi file teks	46
Gambar 4.7. Tampilan hasil dari kompresi algoritma MTF Coding	47
Gambar 4.8. Save file dialog untuk menyimpan data algoritma MTF Coding	47
Gambar 4.9. Kompresi berhasil disimpan	48
Gambar 4.10. Tampilan Halaman menu dekompressi	51
Gambar 4.11. Memilih file yang akan didekompresi	51
Gambar 4.12. Tampilan dari data yang akan didekompresi	52
Gambar 4.13. Proses Dekompressi	52
Gambar 4.14. Masukkan nama file yang akan disimpan	53
Gambar 4.15. Hasil Dekompressi Berhasil Disimpan	53
Gambar 4.16. Perbandingan Ratio of Compression dari Artificial Corpus dan Miscellaneous Corpus.....	56
Gambar 4.17. Perbandingan Ratio of Compression Teks Berulang	57
Gambar 4.18. Perbandingan Ratio of Compression abjad.txt dan nama.txt	58

DAFTAR LAMPIRAN

LAMPIRAN 1 LISTING PROGRAM	A-1
LAMPIRAN 2 CURRICULUM VITAE	B-1

BAB 1

PENDAHULUAN

1.1. Latar Belakang

Kemajuan teknologi yang sangat cepat berdampak pada proses pertukaran data dan informasi. Media cetak kini mulai ditinggalkan dan digantikan oleh media digital. Dengan perubahan ini, cara masyarakat mengirimkan data pun ikut berubah.

Teknologi merupakan pendekatan ilmiah untuk mencapai tujuan praktis. Perkembangan teknologi yang semakin praktis memudahkan masyarakat dalam menjalani aktivitas sehari-hari. Contohnya, dalam hal bertukar pesan, kini masyarakat dapat mengirim pesan dengan mudah, yang dulu harus melalui pos dan memerlukan waktu yang cukup lama. Sekarang, dengan adanya surat elektronik (surel), pengiriman pesan menjadi lebih cepat. Namun, penggunaan surel membutuhkan ruang penyimpanan data yang semakin besar seiring waktu, dan file berukuran besar membutuhkan waktu pengiriman yang lebih lama. Oleh karena itu, diperlukan kompresi data sebelum transmisi untuk mengurangi ukuran file sehingga proses pengiriman data menjadi lebih efisien. Kompresi file teks sangat bermanfaat dalam mengurangi biaya penyimpanan dan transfer data, terutama dalam lingkungan bisnis atau organisasi besar. Kompresi data adalah teknik yang digunakan untuk mengurangi ukuran data asli. Teknik ini biasanya diterapkan pada komputer karena setiap simbol yang ditampilkan memiliki nilai bit yang berbeda. Tujuan dari kompresi ini adalah untuk menghemat ruang penyimpanan dengan mengurangi ukuran data. (Wibowo, 2012).

Kompresi data memiliki 4 parameter yang digunakan untuk menentukan kemampuan dari suatu algoritma yaitu *Compression Ratio* (CR), *Space Saving* (SS), *Bit Rate* dan *Time Process* (Salomon, 2007). Terdapat 2 jenis kompresi data yang digunakan saat ini, yaitu kompresi metode *Lossless* dan kompresi metode *lossy*. *Lossless data compression* adalah sekumpulan algoritma kompresi yang memungkinkan data asli dapat dipulihkan sepenuhnya dari data yang telah dikompresi. Di sisi lain, *lossy compression* adalah metode kompresi data yang setelah proses dekompresi, data yang

diperoleh mungkin tidak persis sama dengan aslinya, tetapi perbedaannya cukup kecil dan mendekati data asli. (Mansyuri, 2021). Untuk melakukan perbandingan algoritma, penulis akan menggunakan kompresi data tipe *Lossless*.

Algoritma Sequitur adalah algoritma rekursif yang dikembangkan oleh Craig Nevill-Manning dan Ian H. Witten pada tahun 1997. Algoritma Sequitur merupakan algoritma waktu linear yang menyimpulkan tata Bahasa bebas konteks (*context-free grammar*) ke dalam suatu kompresi untuk mengurangi masukan yang berulang. Algoritma Sequitur akan mencari suku kata yang sama yang akan diganti menjadi karakter baru yang lebih singkat dengan tujuan untuk mengurangi penggunaan data. Algoritma Sequitur terdiri dari 2 aturan yang berlaku, yaitu Diagram Keunikan (*Uniqueness Diagram*) dan Aturan Kegunaan (*Utility Rule*) (Ervin, 2007).

Uniqueness Diagram mempunyai arti bahwa tidak ada pasangan dari simbol / diagram muncul lebih dari sekali dalam sebuah tata bahasa. Jika hal ini terjadi maka akan melanggar aturan batasan pertama (*Uniqueness Diagram*) sehingga akan membentuk aturan baru (simbol non-terminal) yang akan menggantikan simbol / diagram yang muncul lebih dari sekali.

Utility rule mempunyai arti bahwa setiap aturan produksi digunakan lebih dari sekali. Dan jika ada aturan yang hanya digunakan sekali maka akan terjadi pelanggaran pada batasan kedua (*utility rule*) sehingga aturan yang hanya dipakai sekali akan dihapus atau dihilangkan.

Move-to-Front (MTF) coding adalah algoritma yang digunakan untuk mengurangi masukan data yang berulang. Fungsi *Move-to-Front (MTF) coding* adalah menggantikan simbol yang berulang ke dalam tumpukan indeks di “simbol yang baru digunakan”. Simbol yang sering muncul akan digantikan dengan nol, sedangkan simbol yang lama tidak muncul akan diganti dengan besar angka simbol. Dengan demikian pada akhirnya data diubah menjadi urutan bilangan bulat. Jika data menunjukkan banyak perulangan, maka data akan semakin kecil (Darwiyanto, 2015).

Proses *decoding Move-to-Front (MTF)* lebih singkat dan cukup jelas. Proses ini secara teknis seperti proses encoding, tapi lebih intensif memakan waktu. Pada proses ini posisi sebuah simbol dalam daftar simbol setiap alfabet yang nantinya digunakan untuk memecahkan kode simbol. Seperti proses *encode*, Ketentuan daftar simbol dapat

secara lexicographic (atau cara lain yang telah ditentukan). Data yang telah dikodekan menunjukkan posisi simbol diterjemahkan. Setelah simbol dilakukan *decoding*, simbol dipindahkan ke bagian depan daftar.

1.2. Rumusan Masalah

Berdasarkan latar belakang, maka rumusan masalah yang akan diangkat dalam penelitian ini adalah membandingkan tingkat kinerja antara algoritma Sequitur dengan algoritma *Move-to-Front (MTF) Coding* untuk mengetahui algoritma mana yang lebih efisien dalam kompresi *file* teks.

1.3. Batasan Masalah

Batasan-batasan masalah dalam penelitian ini antara lain:

1. Algoritma yang akan dibandingkan kemampuan kinerjanya adalah algoritma Sequitur dan algoritma *Move-to-Front (MTF) Coding*.
2. *Input* atau data uji data yang digunakan berupa *file* teks dalam bentuk .txt.
3. Pada algoritma *Move-to-Front (MTF) Coding*, yang digunakan hanya Basic Move-to-Front.
4. Parameter yang digunakan dalam pengukuran kinerja kompresi datanya adalah *Compression Ratio* (CR), *Ratio of Compression* (RC), *Space Savings* (SS), dan *Running Time* (RT).
5. Program yang akan dibuat berbentuk aplikasi desktop dengan bahasa pemrograman yang digunakan adalah C#.
6. Pada algoritma Sequitur penulis melakukan kompresi secara manual dikarenakan keterbatasan penulis dalam membuat program Algoritma Sequitur.

1.4. Tujuan Penelitian

Berdasarkan rumusan masalah di atas, maka tujuan dari penelitian ini adalah sebagai berikut:

1. Menerapkan algoritma Sequitur dan algoritma *Move-to-Front* (MTF) *Coding* untuk melakukan kompresi *file* teks.
2. Melakukan perbandingan kinerja algoritma Sequitur dan algoritma *Move-to-Front* (MTF) *Coding* dalam melakukan kompresi file teks dengan merujuk pada beberapa parameter yang menjadi tolok ukur yaitu *Compression Ratio* (CR), *Space Savings* (SS), *Bit Rate* dan *Compression Time*.

1.5. Manfaat Penelitian

Manfaat yang diperoleh dari penelitian ini adalah:

1. Mengetahui algoritma mana yang lebih efisien dalam memperkecil ukuran file teks antara algoritma Sequitur dan algoritma *Move-to-Front* (MTF) *Coding*.
2. Memahami cara kerja dari algoritma Sequitur dan algoritma *Move-to-Front* (MTF) *Coding*.
3. Dapat menghemat ukuran ruang penyimpanan ketika data berhasil dikompresi
4. Hasil yang diperoleh dari perbandingan kinerja kedua algortima diharapkan dapat menjadi bahan referensi untuk penelitian – penelitian selanjutnya dengan masalah terkait.

1.6. Penelitian Relevan

Penelitian terdahulu yang penulis jadikan acuan dalam penggerjaan penelitian ini antara lain adalah sebagai berikut:

1. Penelitian tahun 2022 pada Journal of Informatic and Electronics yang ditulis oleh JL. Ompungsunggu dengan judul Penerapan Kombinasi Algoritma Sequitur Dan Punctured Elias Code Untuk Kompresi File Teks menghasilkan kedua algoritma tersebut berhasil mengurangi ukuran dari file teks (Juni Leuwarta Ompusunggu, 2010).

2. Penelitian tahun 2022 pada Journal of Computing and Informatics Research yang ditulis oleh TF Silaban dengan judul Kompresi File Teks dengan Menggunakan Kombinasi Squitur dan Elias Gamma Code menghasilkan Kelebihan algoritma Sequitur adalah dengan mengganti 2 karakter berdampingan dengan 1 karakter terminal pada karakter set, sedangkan kelebihan algoritma elias gamma code adalah mengganti bit awal karakter dengan bit code elias gamma (Silaban, 2022).
3. Penelitian tahun 2019 pada Jurnal Sistem Informasi yang ditulis oleh Y Darnita dan K. Khairunnisyah dengan judul Kompresi Data Teks Dengan Menggunakan Algoritma Sequitur. Penelitian ini menghasilkan File hasil kompresi oleh algoritma Sequitur memudahkan dalam menggunakan internet sehingga waktu yang diperlukan akan menjadi lebih pendek dan kemungkinan pekerjaan Download dan Upload gagal akan menjadi lebih kecil. Kemudian tertransfer file melalui jaringan akan lebih cepat, waktu pengiriman tergantung dari provider yang cepat atau tidak serta ukuran file yang akan dikirim dan membantu dalam mengurangi ukuran dari file sehingga dapat mengurangi kapasitas penyimpanan suatu memori / RAM.
4. Penelitian tahun 2019 pada Jurnal Terapan Teknologi yang ditulis oleh SM Sunaryo, L Chrisantyo dan Y Lukito dengan judul Analisis Algoritma MTF, MTF-1 dan MTF-2 pada Burrows Wheeler Compression Algorithm. Penelitian ini menghasilkan Pengujian kompresi dan dekompresi pada data Alkitab maupun Calgary Corpus berhasil dilakukan dan menunjukkan MTF-1 mampu memberikan rasio kompresi yang lebih baik dikarenakan jumlah total tiap bit pada proses Huffman lebih sedikit dibandingkan dua metode lainnya (Sunaryo et al., 2019).
5. Penelitian tahun 2012 pada Jurnal Teknologi dan Informasi yang ditulis oleh AD Mahendra, E Suryani, A Aziz dengan judul Analisis Perbandingan Kinerja Kombinasi Algoritma BWT-RLE-MTF-Huffman Dan BWT-MTF-RLE-Huffman Pada Kompresi File. Penelitian ini menghasilkan Hasil pengujian didapat rata-rata total rasio kompresi, waktu kompresi, dan waktu dekompresi untuk BMRH berturut-turut 70.44 %, second/byte, dan, sedangkan BRMH berturut-turut 66.57 %, second/byte, dan second/byte.

1.7. Metodologi Penelitian

Metode yang dilakukan dalam penelitian yang akan dilakukan antara lain:

1. Menentukan Rumusan Masalah

Penulis menentukan permasalahan umum yang akan diangkat dalam penelitian tugas akhirnya.

2. Studi Literatur

Penulis melakukan studi literatur mengenai pemecahan masalah yang dirumuskan sebelumnya, cara penyelesaiannya serta mempelajari hal-hal yang bersangkutan terhadap penyelesaian masalah tersebut.

3. Analisis dan Perancangan

Penulis menganalisis cara penyelesaian masalah tersebut dengan hasil studi literatur. Menentukan algoritma yang akan digunakan serta merancang sistem yang bertujuan untuk menyelesaikan permasalahan di atas.

4. Implementasi

Penulis mengimplementasikan hasil rancangan yang telah dibuat dalam bentuk program atau aplikasi yang dapat dijalankan.

5. Pengujian

Penulis menguji program atau sistem yang telah dibuat apakah berfungsi dengan baik sesuai dengan yang dirancang, serta dapat menyelesaikan masalah yang diangkat dengan benar.

6. Dokumentasi

Penulis menulis laporan dokumentasi hasil penelitian. Laporan tersebut berisi hasil dari tahapan-tahapan yang dilakukan sebelumnya dalam penelitian tersebut dan bertujuan sebagai pedoman untuk penelitian-penelitian selanjutnya.

1.8. Sistematika Penulisan

Berikut ini merupakan sistematika dalam penulisan skripsi ini:

1. BAB 1 Pendahuluan

Bab pertama dalam penulisan skripsi ini merupakan pendahuluan yang membahas mengenai latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, penelitian yang relevan, metodologi penelitian serta sistematika penulisan pada penelitian yang dilakukan.

2. BAB 2 Landasan Teori

Bab kedua berisi penjelasan mengenai teori dasar serta konsep yang akan berhubungan dengan penelitian yang dilakukan, terutama mengenai algoritma yang digunakan dalam penelitian ini yaitu algoritma *Sequitur* dan *algoritma Move-to-Front (MTF) Coding*.

3. BAB 3 Analisis dan Perancangan Sistem

Pada bab ketiga akan diisi dengan analisis terhadap permasalahan yang diangkat dalam penelitian ini, cara penyelesaiannya, serta rancangan kerja dari sistem yang akan dibuat dalam menyelesaikan permasalahan tersebut.

4. BAB 4 Implementasi dan Pengujian Sistem

Bab keempat berisi hasil implementasi atau penggerjaan sistem yang telah dibuat sebelumnya. Bab ini juga berisi hasil pengujian dari sistem tersebut apakah berfungsi dengan baik atau tidak serta hal-hal apa saja yang ditemukan penulis setelah penggerjaan sistem tersebut, baik merupakan kelebihan maupun kekurangan masing-masing algoritma yang diuji.

5. BAB 5 Kesimpulan dan Saran

Bab kelima berisi kesimpulan serta ringkasan yang didapatkan dari hasil penggerjaan skripsi yang dimulai dari pembahasan mengenai masalah hingga pengaplikasian sistem dan pengujinya, serta saran bagi penelitian yang berkaitan kedepannya sebagai bahan acuan ataupun referensi.

BAB 2

LANDASAN TEORI

2.1. Kompresi Data

2.1.1. Pengertian Kompresi Data

Kompresi data adalah proses mengurangi ukuran atau jumlah bit yang diperlukan untuk merepresentasikan informasi digital tanpa mengorbankan informasi yang penting. Tujuannya adalah untuk menghemat ruang penyimpanan dan/atau mengurangi waktu yang dibutuhkan untuk mentransfer data melalui jaringan.

Seiring dengan perkembangannya, teknologi data dari file-file terus berkembang dan mengalami peningkatan ukuran menjadi semakin besar dari waktu ke waktu. Perkembangan teknologi yang semakin maju dan penambahan jumlah pengguna komputer yang semakin banyak menyebabkan perpindahan data dari satu perangkat ke perangkat lain. Data-data tersebut umumnya dikompresi terlebih dahulu agar proses pertukaran tidak memakan waktu yang terlalu lama (Cahayati et al., 2022).

Kompresi data, atau yang sering disebut sebagai teknik pemampatan data, merupakan cara untuk mengurangi ukuran data asli menjadi ukuran yang lebih kecil, yang disebut sebagai data terkompresi. Teknik ini umumnya diterapkan dalam domain komputasi karena setiap simbol yang dihasilkan oleh perangkat komputer memiliki representasi dalam bentuk sejumlah bit yang berbeda. Sebagai contoh, dalam standar ASCII, setiap simbol direpresentasikan dengan 8 bit. Misalnya, kode ASCII untuk huruf A adalah 65, yang dalam representasi biner adalah 01000001. Teknik kompresi data digunakan untuk mengurangi jumlah bit yang dihasilkan dari setiap simbol, dengan harapan dapat mengurangi ukuran data yang dibutuhkan dalam penyimpanan (Wibowo, 2012).

2.1.2. Tujuan kompresi data

Tujuan kompresi data adalah untuk mengurangi data dengan menggunakan cara tertentu tanpa mengorbankan kualitas informasi yang signifikan. Ada beberapa tujuan utama dari kompresi data:

1. Menghemat Ruang Penyimpanan: Salah satu tujuan utama dari kompresi data adalah untuk mengurangi jumlah ruang penyimpanan yang dibutuhkan untuk menyimpan file atau informasi digital. Dengan mengurangi ukuran file, kompresi data memungkinkan pengguna untuk menyimpan lebih banyak data dalam kapasitas penyimpanan yang sama.
2. Mengurangi Biaya Penyimpanan: Dengan mengurangi ruang penyimpanan yang dibutuhkan, kompresi data dapat membantu mengurangi biaya penyimpanan data, terutama dalam skala besar seperti pada perusahaan atau pusat data.
3. Meningkatkan Efisiensi Transfer Data: Kompresi data memungkinkan untuk mengirim atau mentransfer data melalui jaringan dengan lebih efisien. Dengan ukuran yang lebih kecil, data dapat dipindahkan dengan lebih cepat, mengurangi waktu dan sumber daya yang diperlukan untuk mentransfer informasi.
4. Meningkatkan Kinerja Aplikasi: Dengan menggunakan file yang lebih kecil, aplikasi dapat memproses dan mengakses data dengan lebih cepat. Hal ini dapat meningkatkan kinerja aplikasi, terutama pada aplikasi yang memanipulasi data dalam jumlah besar secara berkala.
5. Mengurangi Konsumsi Bandwidth: Dalam konteks komunikasi data, kompresi data membantu mengurangi konsumsi bandwidth dengan mengurangi jumlah data yang harus ditransfer melalui jaringan. Ini dapat menjadi faktor penting dalam situasi di mana bandwidth terbatas atau mahal.

2.2. Dekompreksi Data

2.2.1. Pengertian Dekompreksi Data

Dekompreksi adalah proses yang digunakan untuk mengembalikan data yang telah dikompresi menjadi bentuk aslinya. Jika hasil dekompreksi menghasilkan data yang identik dengan data aslinya sebelum kompresi, maka proses tersebut disebut sebagai kompresi tanpa kehilangan informasi (lossless compression). Sebaliknya, jika hasil dekompreksi menghasilkan data yang tidak identik dengan data aslinya karena beberapa

informasi dihilangkan, meskipun informasi penting tetap terjaga, maka proses tersebut disebut sebagai kompresi dengan kehilangan informasi (lossy compression). Setelah proses kompresi dilakukan pada sebuah gambar atau file, citra tersebut dapat dikembalikan ke bentuk aslinya melalui proses dekompresi (Mahesa, 2017).

2.2.2. Tujuan dekompresi data

Tujuan dari dekompresi adalah untuk mendapatkan kembali informasi asli dari data yang telah dikompresi tanpa kehilangan informasi penting. Proses dekompresi memerlukan penggunaan algoritma yang sesuai dengan yang digunakan dalam proses kompresi. Hal ini karena setiap algoritma kompresi memiliki metode khusus untuk mengkonversi data menjadi kode dan mengurai informasi dalam data tersebut.

2.3. Pengukuran Kompresi

Kompresi data memiliki 4 parameter yang digunakan untuk menentukan kemampuan dari suatu algoritma yaitu Compression Ratio (CR), Space Saving (SS), Bit Rate dan Time Process (Salomon, 2007), antara lain:

2.3.1. Ratio of Compression (RC)

Merupakan perbandingan antara ukuran data asli dengan ukuran data yang sudah dikompresi.

$$RC = \frac{\text{Ukuran data asli}}{\text{Ukuran data setelah dikompresi}}$$

2.3.2. Compression Ratio (CR)

Merupakan kebalikan dari *Ratio of Compression*. Yaitu perbandingan antara ukuran data setelah dikompresi dengan ukuran data asli.

$$CR = \frac{\text{Ukuran data setelah dikompresi}}{\text{Ukuran data asli}}$$

2.3.3. Space Savings (SS)

Parameter yang digunakan untuk mengukur besarnya ruang penyimpanan yang dihemat setelah kompresi. Space Saving merupakan persentase dari ruang yang dihemat dari ukuran data aslinya.

$$SS = 100\% - \frac{Ukuran\ data\ asli}{Ukuran\ data\ setelah\ dikompresi} \times 100\%$$

2.3.4. Running Time

Merupakan waktu yang digunakan sistem dalam proses kompresi data. Semakin sedikit waktu yang diperlukan, maka semakin efisien teknik kompresi yang digunakan.

2.4. Teknik Kompresi

2.4.1. Jenis teknik kompresi data

Terdapat 2 jenis teknik kompresi data (Mansyuri, 2021), antara lain:

1. *Lossless*, Kompresi Lossless adalah suatu teknik kompresi data di mana data yang dikompresi dapat dipulihkan sepenuhnya tanpa kehilangan informasi. Dengan kata lain, setelah data dikompresi dan kemudian didekompresi, hasilnya identik dengan data asli. Proses kompresi Lossless dirancang sedemikian rupa sehingga tidak ada kerugian data selama proses tersebut.

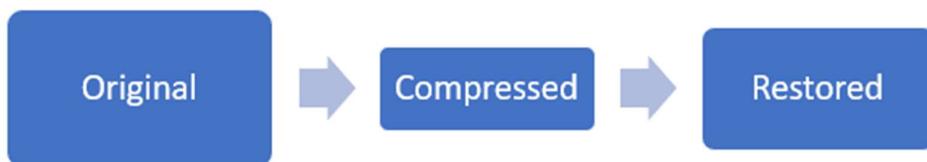
Tujuan dari kompresi Lossless adalah untuk mengurangi ukuran file atau data tanpa menghilangkan informasi yang terkandung di dalamnya. Teknik ini sangat berguna dalam situasi di mana kualitas dan integritas data sangat penting, seperti pada file teks, dokumen, file program, atau file gambar yang memerlukan reproduksi yang setara dengan data asli.



Gambar 2.1 Proses Kompresi dan Dekompresi Data Lossless

2. *Lossy*, Kompresi lossy adalah suatu teknik kompresi data di mana beberapa informasi dari data asli dihilangkan untuk menghasilkan versi yang lebih ringkas. Dalam proses kompresi lossy, sejumlah kerugian kualitas atau detail diizinkan demi mengurangi ukuran file. Setelah data mengalami kompresi lossy, tidak mungkin untuk mengembalikan data ke bentuk asli dengan presisi sempurna karena informasi yang hilang tidak dapat dipulihkan sepenuhnya.

Teknik kompresi lossy sering diterapkan pada data seperti file gambar, audio, dan video. Keuntungan dari kompresi lossy adalah ukuran file yang jauh lebih kecil dibandingkan dengan kompresi Lossless, sehingga memungkinkan penghematan ruang penyimpanan dan pengiriman data yang lebih efisien. Namun, kelemahan utamanya adalah adanya degradasi kualitas karena beberapa informasi dihapus selama proses kompresi



Gambar 2.2 Proses Kompresi dan Dekompresi Data Lossy

2.5. Algoritma Sequitur

Menurut Nevill-Manning dan Ian Witlen (1997), Sequitur adalah algoritma waktu linier yang menyimpulkan tata bahasa bebas konteks (context-free grammaar) ke dalam suatu pemampatan untuk mengurangi masukan yang berulang. Cara kerja sequitur terdiri dari pemastian dua sifat yang berlaku. Algoritma Sequitur menjalankan batasan di dalam sebuah tata bahasa yang ketika diagram keunikan (uniqueness diagram) dilanggar, sebuah aturan baru dibentuk, dan ketika batasan aturan kegunaan (utility rule) dilanggar, aturan yang sia-sia dihapus (Simanjuntak, 2021).

2.5.1. Kelebihan Algoritma Sequitur

- Efisiensi: Sequitur adalah algoritma kompresi data yang efisien. Algoritma ini mampu mengenali pola berulang dalam data secara otomatis dan menggantinya dengan aturan produksi baru. Hal ini memungkinkan untuk menghasilkan representasi yang lebih ringkas dari data yang berulang.
- Kemampuan Adaptasi: Sequitur dapat secara adaptif menyesuaikan diri dengan data input yang berbeda. Algoritma ini mampu menemukan struktur yang berulang atau pola yang muncul dalam data tanpa memerlukan pengetahuan sebelumnya tentang data tersebut.
- Kompleksitas Waktu yang Rendah: Algoritma Sequitur memiliki kompleksitas waktu yang rendah, terutama untuk data dengan struktur yang berulang atau memiliki pola tertentu. Ini membuatnya cocok untuk aplikasi di mana efisiensi waktu penting, seperti dalam kompresi data real-time atau aplikasi yang membutuhkan pemrosesan cepat.
- Kompresi Lossless: Sequitur adalah algoritma kompresi data yang bekerja secara lossless, yang berarti bahwa data yang dikompresi dan kemudian didekompresi akan sama persis dengan data aslinya tanpa kehilangan informasi.
- Penerapan yang Luas: Algoritma Sequitur dapat diterapkan dalam berbagai jenis data, termasuk teks, citra, dan data lainnya. Hal ini membuatnya relevan dalam berbagai aplikasi yang membutuhkan kompresi data yang efisien.

2.5.2 Kekurangan Algoritma Sequitur

- Kompleksitas Memori: Algoritma Sequitur cenderung memerlukan lebih banyak memori untuk menjalankan proses kompresi dan dekompresi. Ini karena algoritma harus menyimpan aturan produksi yang digunakan untuk merepresentasikan pola berulang dalam data. Seiring dengan pertumbuhan ukuran data yang diproses, kebutuhan akan memori juga dapat meningkat secara signifikan.
- Waktu Proses yang Lambat: Meskipun Sequitur memiliki kompleksitas waktu yang rendah untuk data dengan struktur yang berulang, namun untuk data yang tidak memiliki pola berulang atau memiliki pola yang kompleks, algoritma ini dapat mengalami kinerja yang lambat. Proses pencarian pola dan pembentukan aturan produksi baru dapat menjadi lebih sulit dan memakan waktu.
- Keterbatasan dalam Kompresi Data Tertentu: Sequitur mungkin tidak efektif untuk semua jenis data. Algoritma ini terutama efektif untuk data yang memiliki pola berulang atau struktur hierarkis yang dapat diidentifikasi dan direpresentasikan dengan aturan produksi. Untuk data yang lebih acak atau tidak memiliki struktur yang jelas, Sequitur mungkin tidak memberikan kompresi yang signifikan.
- Keterbatasan Implementasi: Implementasi algoritma Sequitur dapat menjadi rumit. Pembentukan dan manajemen aturan produksi serta pencarian pola dalam data memerlukan pemahaman yang mendalam tentang algoritma. Hal ini dapat membuat pengembangan dan pemeliharaan sistem yang menggunakan algoritma Sequitur menjadi lebih sulit.

2.5.3. Cara kerja algoritma sequitur

Berikut adalah cara kerja algoritma sequitur:

A. Proses Kompresi

Algoritma Sequitur bekerja dengan cara mengenali pola berulang atau struktur hierarkis dalam data dan menggantinya dengan aturan produksi baru. Berikut adalah langkah-langkah umum cara kerja algoritma Sequitur:

1. Pemrosesan Data: Data input dibagi menjadi token atau simbol-simbol yang merepresentasikan elemen data seperti karakter dalam teks, piksel dalam gambar, atau item dalam rangkaian data lainnya.
2. Pencarian Pola Berulang: Algoritma Sequitur memindai data input untuk mengidentifikasi pola berulang atau struktur hierarkis yang muncul secara konsisten dalam data.
3. Pembentukan Aturan Produksi: Setelah pola berulang ditemukan, aturan produksi baru dibentuk untuk merepresentasikan pola tersebut secara singkat. Aturan ini mencatat pola berulang dan cara mereka saling berhubungan, sehingga data dapat direkonstruksi kembali dengan menggunakan aturan ini.
4. Penggantian Pola dengan Aturan Produksi: Pola berulang yang telah ditemukan digantikan dengan aturan produksi yang sesuai. Ini mengurangi jumlah data yang sebenarnya perlu disimpan dalam representasi kompresi, menghasilkan ukuran file yang lebih kecil.
5. Penyimpanan Aturan Produksi: Aturan produksi yang baru dibentuk disimpan dan dipertahankan untuk digunakan dalam proses dekompresi.

B. Proses Dekompresi

Berikut adalah langkah-langkah kerja proses dekompreksi algoritma Sequitur:

1. Rekonstruksi Data dari Aturan Produksi: Data yang telah dikompresi diambil, dan aturan produksi yang digunakan dalam proses kompresi diterapkan untuk mengembalikan data ke bentuk aslinya.
2. Penggantian Aturan dengan Pola: Aturan produksi digunakan untuk mengganti simbol-simbol dalam data kompresi dengan pola yang sesuai seperti yang tercatat dalam aturan tersebut.
3. Penggabungan Pola: Pola yang dihasilkan oleh penggantian aturan digabungkan untuk menghasilkan data yang telah didekompresi secara lengkap.
4. Rekonstruksi Data Asli: Data yang telah didekompresi kemudian diproses lebih lanjut, jika diperlukan, untuk mengembalikannya ke bentuk aslinya sebelum kompresi.

2.5.4. Contoh Perhitungan Sequitur

Berikut String : “DONISIUS MARTIN SIRAIT”

Seleksi teks : ‘DO’, ‘ON’, ‘NI’, ‘IS’, ‘SI’, ‘IU’, ‘US’, ‘S
 spasi’, ‘spasi M’, ‘MA’, ‘AR’, ‘RT’, ‘TI’,
 ‘IN’, ‘N spasi’, ‘spasi S’, ‘SI’, ‘IR’, ‘RA’,
 ‘AI’, ‘IT’

Perulangan terjadi pada ‘SI’, maka diganti dengan simbol non-terminal ‘B’ → ‘SI’.

Perubahan string : “DONIBUS MARTIN BRAIT”

dilakukan seleksi teks Kembali : ‘DO’, ‘ON’, ‘NI’, ‘IS’, ‘BU’, ‘US’, ‘S
 spasi’, ‘spasi M’, ‘MA’, ‘AR’, ‘RT’,
 ‘TI’, ‘IN’, ‘N spasi’, ‘spasi S’, ‘BR’,
 ‘RA’, ‘AI’, ‘IT’

Setelah dilakukan seleksi teks kembali, tidak ada ditemukan simbol non-terminal yang sekali pakai, dan tidak ada perulangan pasangan kata. Sehingga proses kompresi selesai.

Maka, hasil kompresi : “DONIBUS MARTIN BRAIT”

Contoh dalam bentuk tabel:

Tabel 2.1. Contoh Proses Kompresi Sequitur

String	Simbol Berulang	Rule	Perubahan String	Simbol yang Dihapus
DONISIUS MARTIN SIRAIT	SI	B → SI	DONIBUS MARTIN BRAIT	-
DONIBUS MARTIN BRAIT	-	-	-	-

Untuk mengetahui ukuran teks yang akan dikompresi, terlebih dahulu diurutkan berdasarkan frekuensi kemunculan karakter terbanyak hingga terkecil, dan menghitung bit dari setiap karakternya. Berikut adalah perhitungan dalam bentuk tabel:

Tabel 2.2. Contoh Proses Perhitungan Kompresi Biasa

KARAKTER	FREKUENSI	DESIMAL	BINER	BIT	BIT X FREKUENSI
I	5	73	1001001	8	40
S	3	83	1010011	8	24
SPASI	2	32	100000	8	16
A	2	65	1000001	8	16
N	2	78	1001110	8	16
R	2	82	1010010	8	16
T	2	84	1010100	8	16
D	1	68	1000100	8	8
M	1	77	1001101	8	8
O	1	79	1001111	8	8
U	1	85	1010101	8	8
Total	22	-	-	-	176

Berikut setelah dilakukan kompresi “DONIBUS MARTIN BRAIT”:

Tabel 2.3. Contoh Proses Perhitungan Kompresi Algoritma Sequitur

KARAKTER	FREKUENSI	DESIMAL	BINER	BIT	BIT X FREKUENSI
I	3	73	1001001	8	24
S	1	83	1010011	8	8
SPASI	2	32	100000	8	16
A	2	65	1000001	8	16
B	2	66	1000010	8	16
N	2	78	1001110	8	16
R	2	82	1010010	8	16
T	2	84	1010100	8	16
D	1	68	1000100	8	8
M	1	77	1001101	8	8
O	1	79	1001111	8	8
U	1	85	1010101	8	8
Total	20	-	-	-	160

Perhitungan kompresi:

$$\text{Ratio of Compression (RC)} = 22 / 20 = 1.1$$

$$\text{Compression Ratio (CR)} = 20 / 22 = 0.91$$

$$\text{Space Savings (SS)} = 100\% - 91\% = 9\%$$

Dekompresi

Pada saat dilakukan kompresi rule akan disimpan. Untuk melakukan dekompresi rule yang telah disimpan akan digunakan kembali untuk mengembalikan string ke bentuk semula, seperti pada tabel dibawah:

Tabel 2.4. Contoh Dekompresi Algoritma Sequitur

String	Rule	Keterangan	Perubahan String
DONIBUS MARTIN BRAIT	$B \rightarrow SI$	Simbol B akan menggantikan kata 'SI' ke string awal.	DONISIUS MARTIN SIRAIT

Pada proses dekompresi dilakukan secara berurutan dari rule terakhir sampai rule awal yang akan mengembalikan string kembali ke awal sebelum dikompresi.

2.6. Algoritma Move-to-Front (MTF) Coding

Algoritma *Move-to-Front (MTF) Coding* adalah sebuah metode kompresi data yang memanipulasi urutan karakter dalam sebuah *string*. Cara kerjanya adalah dengan menggeser karakter yang baru saja diakses ke posisi awal alfabet, sehingga karakter tersebut akan lebih cepat diakses pada saat berikutnya. Misalnya, ketika sebuah karakter diakses, karakter tersebut akan dipindahkan ke posisi pertama dalam alfabet, diikuti oleh penggeseran karakter lainnya sesuai urutan abjad.

Proses ini memungkinkan untuk menciptakan pola yang unik dalam urutan karakter, sehingga karakter yang sering muncul akan berada di dekat awal string. Sebagai contoh, jika sebuah karakter sering muncul dalam sebuah string, setelah beberapa kali akses, karakter tersebut akan berada di depan string. Hal ini dapat meningkatkan efisiensi kompresi karena karakter yang sering muncul akan memiliki representasi yang lebih pendek.

MTF Coding juga digunakan dalam kombinasi dengan metode kompresi lainnya, seperti Huffman Coding, untuk meningkatkan efisiensi kompresi. Dalam

proses dekompresi, langkah-langkah yang sama diterapkan untuk mengembalikan urutan karakter ke posisi awalnya sebelum proses kompresi dilakukan.

2.6.1. Kelebihan Move-to-Front (MTF) Coding

Kelebihan dari algoritma *Move-to-Front (MTF) Coding* antara lain adalah sebagai berikut:

1. Sederhana: MTF *Coding* adalah metode yang relatif sederhana dalam implementasinya. Konsepnya mudah dipahami dan dapat diterapkan dengan mudah dalam berbagai aplikasi kompresi data.
2. Efektif untuk Data Berulang: *MTF Coding* efektif dalam mengompresi data yang memiliki pola berulang, di mana karakter yang sama muncul secara berulang dalam urutan data. Dengan menggeser karakter yang baru diakses ke depan, karakter yang sering muncul akan memiliki representasi yang lebih pendek, menghasilkan kompresi yang lebih baik.
3. Adaptif: MTF *Coding* dapat secara adaptif menyesuaikan diri dengan karakteristik data yang berbeda. Ini berarti bahwa algoritma ini dapat bekerja dengan baik dalam berbagai jenis data, termasuk data teks, audio, atau gambar.

2.6.2. Kekurangan Algoritma Move-to-Front (MTF)

Kekurangan dari algoritma *Move-to-Front (MTF)* adalah sebagai berikut ini:

1. Tidak Efektif untuk Data Acak: MTF *Coding* kurang efektif saat digunakan pada data yang acak atau memiliki sedikit pola berulang. Dalam kasus ini, penggunaan MTF Coding mungkin tidak memberikan kompresi yang signifikan atau bahkan dapat memperbesar ukuran data.
2. Sensitif terhadap Pengaturan Awal: Efektivitas MTF *Coding* dapat dipengaruhi oleh pengaturan awal alfabet atau urutan karakter. Jika pengaturan awal tidak sesuai dengan karakteristik data yang sebenarnya, kompresi yang dihasilkan mungkin tidak optimal.

3. Penanganan Karakteristik Data yang Spesifik: MTF *Coding* mungkin tidak optimal untuk beberapa jenis data yang memiliki karakteristik tertentu. Misalnya, untuk data dengan banyak variasi karakter dan sedikit pola berulang, MTF *Coding* mungkin tidak memberikan kompresi yang baik.
4. Penggunaan Memori yang Lebih Besar: Proses MTF *Coding* dapat memerlukan penyimpanan tambahan untuk mengelola posisi karakter dalam alfabet. Ini dapat meningkatkan kebutuhan akan memori, terutama untuk data dengan panjang yang besar.

2.6.3. Langkah Algoritma Move-to-Front (MTF)

A. Kompresi

Berikut adalah beberapa langkah dasar dalam proses kompresi menggunakan algoritma *Move-to-Front (MTF)*:

- Inisialisasi Alfabet: Langkah pertama dalam proses kompresi adalah menginisialisasi alfabet, yaitu urutan karakter yang akan digunakan oleh algoritma MTF. Biasanya, alfabet diatur dalam urutan alfabet yang standar.
- Iterasi melalui Data: Data input diiterasi karakter per karakter. Setiap karakter diubah ke indeksnya dalam alfabet.
- Penggeseran Karakter: Setelah mengonversi karakter ke indeks alfabet, karakter tersebut dipindahkan ke posisi awal alfabet. Misalnya, jika karakter "c" muncul, karakter tersebut dipindahkan ke posisi pertama dalam alfabet.
- Output Indeks: Setelah karakter dipindahkan, indeks karakter tersebut dalam alfabet menjadi output dari proses kompresi.
- Memperbarui Alfabet: Setelah karakter diproses, alfabet diperbarui untuk mempertahankan urutan karakter yang baru.
- Mengulangi Langkah 2-5: Proses ini diulangi untuk setiap karakter dalam data input.

- Output: Output dari proses kompresi adalah urutan indeks karakter yang direpresentasikan menggunakan algoritma MTF.

B. Dekompresi

Berikut adalah beberapa langkah dasar dalam proses dekompresi menggunakan algoritma *Move-to-Front (MTF)*:

1. Inisialisasi Alfabet: Dekompresi dimulai dengan menginisialisasi alfabet yang sama dengan yang digunakan dalam proses kompresi.
2. Iterasi melalui Indeks: Indeks karakter yang dikompresi diiterasi satu per satu.
3. Penggeseran Karakter: Untuk setiap indeks, karakter yang sesuai dengan indeks tersebut dalam alfabet dipindahkan ke posisi awal alfabet.
4. Output Karakter: Karakter yang dipindahkan menjadi output dari proses dekompresi.
5. Memperbarui Alfabet: Alfabet diperbarui untuk mempertahankan urutan karakter yang baru.
6. Mengulangi Langkah 2-5: Proses ini diulangi untuk setiap indeks dalam data yang dikompresi.
7. Output: Output dari proses dekompresi adalah urutan karakter yang direpresentasikan dalam bentuk aslinya sebelum dikompresi menggunakan algoritma MTF.

2.6.4. Contoh perhitungan algoritma Move-to-Front (MTF)

String : “DONISIUS MARTIN SIRAIT”

```
Input : ['01000100', '01001111', '01001110', '01001001',
        '01010011', '01001001', '01010101', '01010011',
        '00100000', '01001101', '01000001', '01010010',
        '01010100', '01001001', '01001110', '00100000',
        '01010011', '01001001', '01010010', '01000001',
        '01001001', '01010100']
```

1. Lakukan inisiasi tabel alfabet

Tabel diinisiasi dengan urutan alfabet berikut ini:

[Spasi, A, D, I, M, N, O, R, S, T, U]

2. Proses Kompresi:

Tabel 2.5. Contoh Perhitungan Algoritma MTF CODING

Input	Output (indeks huruf)	Tabel Setelah Proses
D	2	[D, Spasi, A, I, M, N, O, R, S, T, U]
O	6	[O, D, Spasi, A, I, M, N, R, S, T, U]
N	6	[N, O, D, Spasi, A, I, M, R, S, T, U]
I	5	[I, N, O, D, Spasi, A, M, R, S, T, U]
S	8	[S, I, N, O, D, Spasi, A, M, R, T, U]
I	1	[I, S, N, O, D, Spasi, A, M, R, T, U]
U	10	[U, I, S, N, O, D, Spasi, A, M, R, T]
S	2	[S, U, I, N, O, D, Spasi, A, M, R, T]
Spasi	6	[Spasi, S, U, I, N, O, D, A, M, R, T]
M	8	[M, Spasi, S, U, I, N, O, D, A, R, T]
A	8	[A, M, Spasi, S, U, I, N, O, D, R, T]
R	9	[R, A, M, Spasi, S, U, I, N, O, D, T]
T	10	[T, R, A, M, Spasi, S, U, I, N, O, D]
I	7	[I, T, R, A, M, Spasi, S, U, N, O, D]
N	8	[N, I, T, R, A, M, Spasi, S, U, O, D]
Spasi	6	[Spasi, N, I, T, R, A, M, S, U, O, D]
S	7	[S, Spasi, N, I, T, R, A, M, U, O, D]
I	3	[I, S, Spasi, N, T, R, A, M, U, O, D]
R	5	[R, I, S, Spasi, N, T, A, M, U, O, D]
A	6	[A, R, I, S, Spasi, N, T, M, U, O, D]

I	2	[I, A, R, S, Spasi, N, T, M, U, O, D]
T	6	[T, I, A, R, S, Spasi, N, M, U, O, D]

3. Output

Output kompresi = [2, 6, 6, 5, 8, 1, 10, 2, 6, 8, 8, 9, 10, 7,
8, 6, 7, 3, 5, 6, 2, 6]

Output kode = [T, I, A, R, S, Spasi, N, M, U, O, D]

Jumlah char unik = 11

Ket: Untuk penyimpanan output kompresi dapat dilakukan dalam beberapa bit saja tergantung jumlah alfabet yang ditemukan. Jika alfabet yang ditemukan seperti pada contoh yaitu 10 alfabet dapat disimpan masing-masing dalam 4 bit data saja yaitu (0000 – 1010).

Metadata = 4 bit

Output metadata = 00000100

Output biner = ['0010', '0110', '0110', '0101',
'1000', '0001', '1010', '0010',
'0110', '1000', '1000', '1001',
'1010', '0111', '1000', '0110',
'0111', '0011', '0101', '0110',
'0010', '0110']

Output kode = ['01010100', '01001001', '01000001',
'01010010', '01010011', '00100000',
'01001110', '01001101', '01010101',
'01001111', '01000100']

Selanjutnya adalah menambahkan padding Untuk dapat mengubah data hasil kompresi menjadi bentuk byte, maka ditambahkan padding untuk membuat data tersebut menjadi kelipatan 8. Serta jumlah paddingnya disimpan dalam meta data tertentu agar dapat dikembalikan dalam proses dekompresinya.

Padding = 00000000, maka output biner menjadi:

Output biner = ['00100110', '01100101', '10000001', '10100010',
 '01101000', '10001001', '10100111', '10000110',
 '01110011', '01010110', '00100110', '00000000'].

Output kode = ['01010100', '01001001', '01000001',
 '01010010', '01010011', '00100000',
 '01001110', '01001101', '01010101',
 '01001111', '01000100'].

Kemudian menambahkan Pemisah (delimiter) untuk memisahkan data hasil mtf yang terdiri dari deret angka perpindahan serta dictionary yang berisi daftar byte/char unik yg sudah disusun tersebut, untuk proses pengembalian data nanti.

pemisah = 0xFF / 11111111, maka output akan menjadi:

Output = ['00000100', '00100110', '01100101', '10000001', '10100010',
 '01101000', '10001001', '10100111', '10000110', '01110011',
 '01010110', '00100110', '00000000', '11111111', '01010100',
 '01001001', '01000001', '01010010', '01010011', '00100000',
 '01001110', '01001101', '01010101', '01001111', '01000100']

$$\text{Ratio of Compression (RC)} = 22 / 25 = 0.88$$

$$\text{Compression Ratio (CR)} = 25 / 22 = 1.13$$

$$\text{Space Savings (SS)} = 100\% - 113\% = -13\%$$

BAB 3

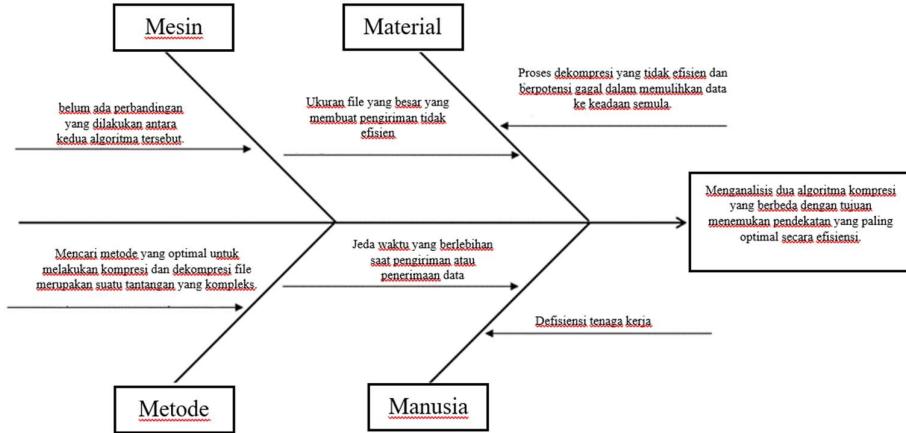
ANALISIS DAN PERANCANGAN SISTEM

3.1. Analisis Sistem

Teknik pemecahan masalah melalui analisis sistem melibatkan langkah-langkah yang terdiri dari memecah masalah menjadi bagian-bagian yang lebih kecil. Tujuannya adalah untuk mempermudah pemahaman, identifikasi, dan penilaian masalah serta kebutuhan sistem, serta mengatasi rintangan yang mungkin timbul dalam pengembangan sistem. Analisis sistem menjadi landasan dalam merancang sistem agar dapat berfungsi sesuai dengan harapan. Proses ini melibatkan langkah-langkah seperti identifikasi masalah, pemahaman, analisis sistem, penerapan, serta pembuatan laporan dan evaluasi. Tahapan identifikasi dijabarkan sebagai berikut:

3.1.1. Analisis Masalah

Analisis masalah bertujuan untuk mengidentifikasi dengan jelas permasalahan yang terjadi, sehingga dapat diatasi dengan sistem yang akan dikembangkan. Contohnya, dalam konteks ini, fokusnya adalah menciptakan sebuah sistem yang dapat membandingkan proses kompresi file teks menggunakan algoritma Sequitur dan algoritma *MTF Coding*. Untuk menggambarkan analisis masalah dalam penelitian ini, akan digunakan diagram Ishikawa (Fishbone Diagram) seperti yang tertera di bawah ini:



Gambar 3.1. Fish Bone Diagram

Dari ilustrasi pada gambar tersebut, tergambar bahwa fokus utama penelitian adalah mengevaluasi efektivitas proses kompresi data terhadap berbagai jenis data, yang membutuhkan perbandingan antara dua algoritma untuk mengetahui perbedaannya. Sementara itu, empat garis diagonal lainnya mewakili dasar-dasar penelitian, yaitu masalah yang meliputi aspek mesin, metode, material, dan manusia.

3.1.2. Analisis Kebutuhan

Analisis kebutuhan melibatkan proses mengidentifikasi sejumlah kebutuhan yang diperlukan untuk pengembangan sistem. Kebutuhan tersebut dapat diklasifikasikan menjadi dua kategori utama: kebutuhan fungsional dan kebutuhan non-fungsional. Kebutuhan fungsional mencakup fitur-fitur yang harus ada dalam sistem, seperti respons terhadap input dan perilaku sistem. Sementara itu, kebutuhan non-fungsional mengacu pada karakteristik sistem, seperti batasan layanan, kinerja waktu dan biaya, standarisasi, batasan pengembangan, antarmuka pengguna, dan lainnya.

1. Kebutuhan Fungsional

Dalam konteks sistem ini, kebutuhan fungsional yang diperlukan mencakup:

- Kemampuan sistem untuk beroperasi tanpa kesalahan,
- Kemampuan sistem untuk membaca *file* teks dengan format .txt,

- c. Kemampuan sistem untuk mengompresi *file* teks menggunakan algoritma Sequitur dan MTF Coding,
- d. Kemampuan sistem untuk menyimpan file hasil kompresi,
- e. Kemampuan sistem untuk melakukan dekompresi terhadap file hasil kompresi,
- f. Kemampuan sistem untuk menampilkan hasil pengukuran kompresi, dan
- g. Kemampuan sistem untuk membandingkan hasil kompresi menggunakan dua algoritma yang telah ditentukan.

2. Kebutuhan Non-fungsional

Kebutuhan non-fungsional yang perlu diperhatikan dalam pengembangan sistem ini meliputi:

- a. Kinerja sistem yang optimal,
- b. Antarmuka pengguna yang intuitif,
- c. Interaktivitas sistem seperti pesan peringatan pop-up untuk masalah yang muncul
- d. Pedoman penggunaan yang jelas dan efisien.

3.1.3. Analisis Proses

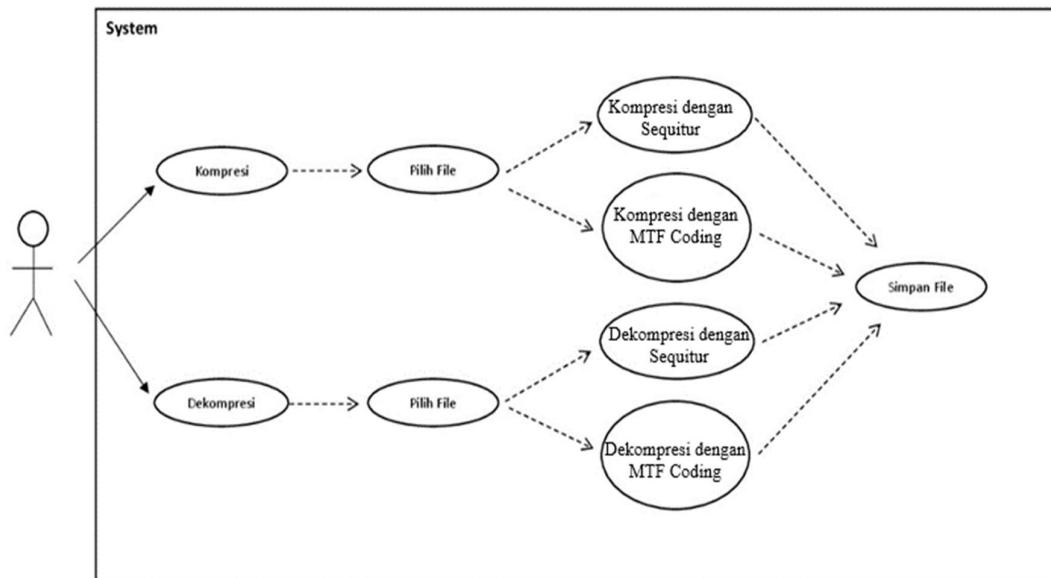
Secara umum, analisis proses melibatkan pemahaman terhadap keseluruhan langkah-langkah yang terjadi dalam sistem yang sedang dikembangkan. Ini mencakup bagaimana pengguna menggunakan sistem, mulai dari pembukaan program hingga penggunaan yang efektif. Dalam sistem ini, pengguna membuka aplikasi dan memilih file untuk dikompresi. Proses kompresi dilakukan menggunakan dua algoritma, yaitu Sequitur dan MTF Coding secara terpisah, untuk mengurangi risiko kesalahan keseluruhan yang mungkin terjadi akibat kegagalan pada salah satu proses algoritma. Setelah berhasil dikompresi, sistem akan menampilkan pengukuran kompresi dan membandingkan hasil kompresi dari kedua algoritma. Selain itu, sistem juga memiliki kemampuan untuk menyimpan dan melakukan dekompresi terhadap file hasil kompresi untuk mengembalikannya ke bentuk aslinya.

3.2. Pemodelan Sistem

Pemodelan sistem melibatkan representasi dari interaksi atau keterkaitan antara berbagai komponen yang akan disertakan dalam sistem yang sedang dikembangkan. Dalam proses pemodelan sistem ini, akan dijelaskan menggunakan tiga jenis diagram, yaitu diagram Use Case, diagram Aktivitas, dan diagram Urutan.

3.2.1. Diagram Use Case

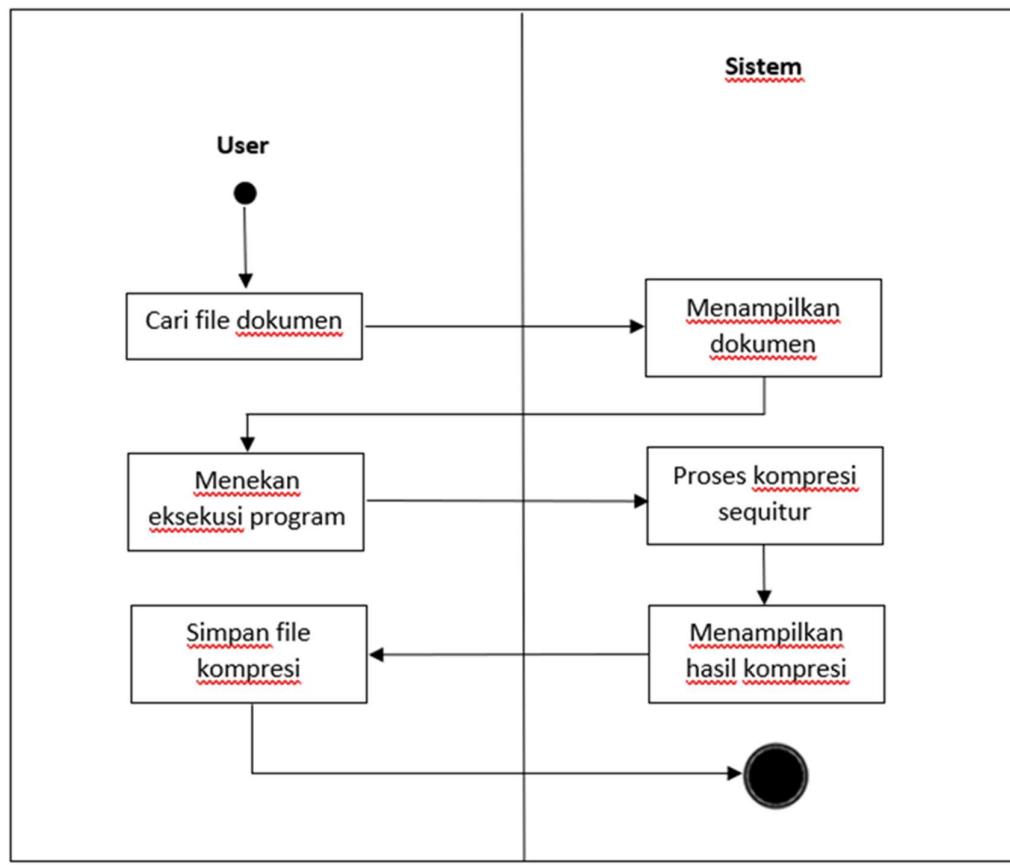
Di dalam diagram Use Case, terdapat penjelasan mengenai keterkaitan antara aktor, yang merupakan entitas yang terlibat dalam proses sistem, dengan komponen-komponen yang menggunakan sistem tersebut.



Gambar 3.2 Use Case Diagram

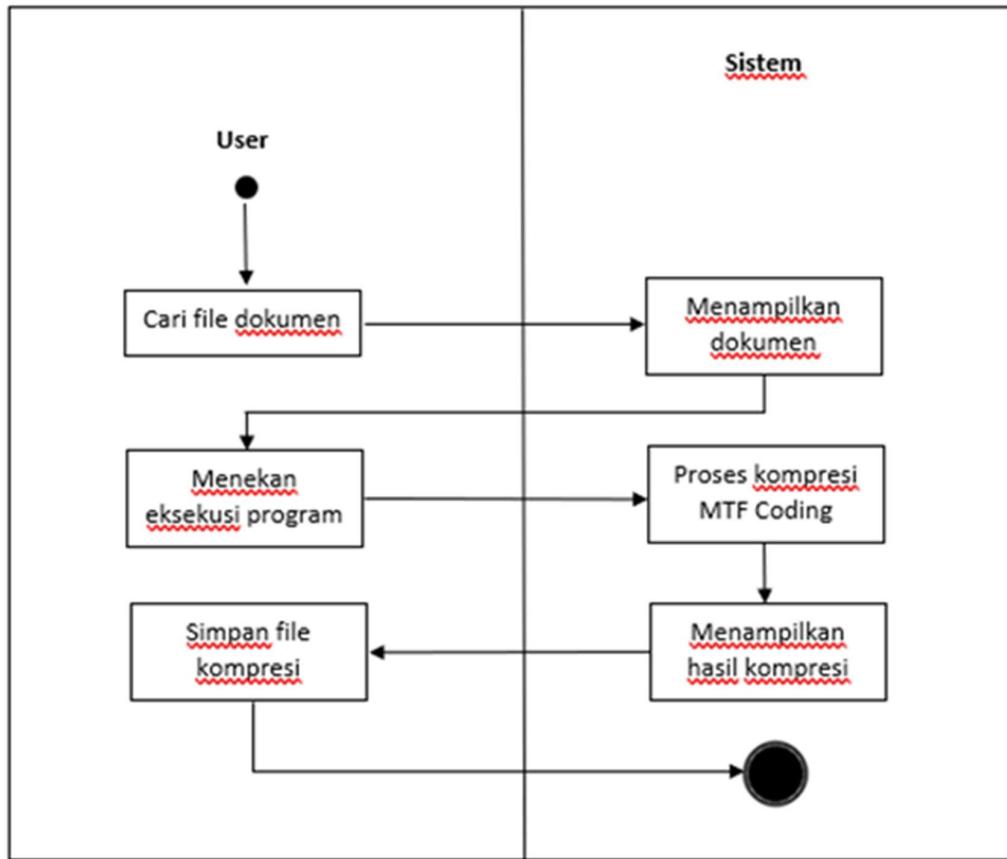
3.2.2. Diagram Activity

Diagram Activity berfungsi untuk memvisualisasikan serangkaian proses yang terjadi antara pengguna dan sistem, serta mengilustrasikan interaksi antara keduanya.



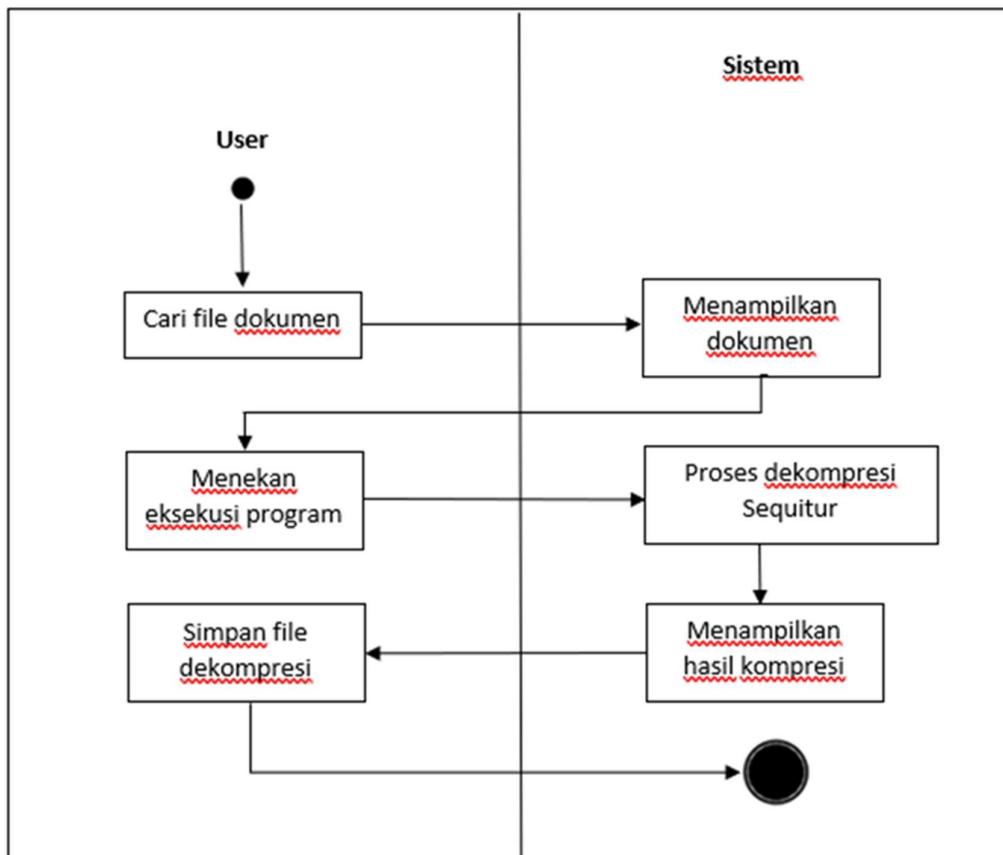
Gambar 3.3. Activity Diagram Proses Kompresi Algoritma Sequitur

Pada Activity Diagram ini, tahap pertama adalah pengguna memasukkan file yang akan dikompres menggunakan algoritma Sequitur. Setelah penginputan dilakukan, sistem akan menampilkan isi file tersebut. Selanjutnya, pengguna akan menjalankan sistem untuk melakukan kompresi menggunakan algoritma Sequitur. Setelah proses kompresi selesai, hasilnya akan ditampilkan dan pengguna dapat menyimpan data hasil kompresi tersebut.



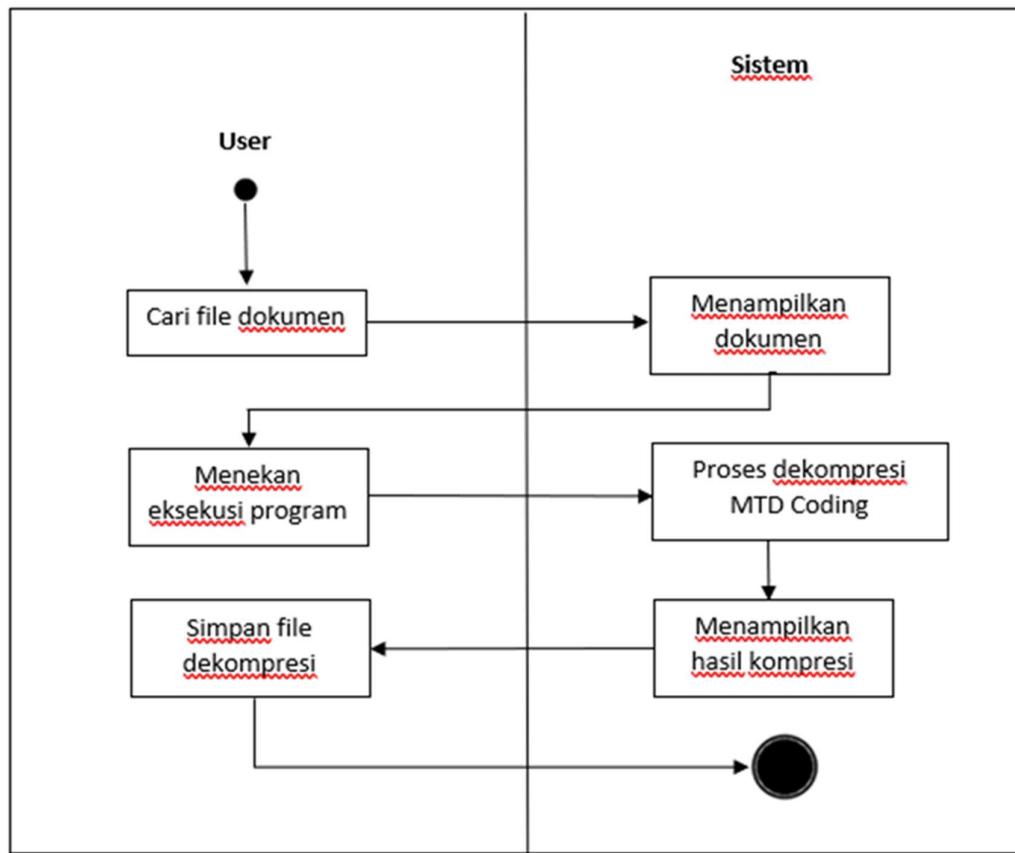
Gambar 3.4. Activity Diagram Proses Kompresi Algoritma MTF Coding

Seperti yang terlihat pada Activity Diagram dalam proses kompresi menggunakan algoritma Sequitur, Diagram Activity dalam penerapan algoritma MTF Coding juga dimulai dengan langkah peng-input-an file yang akan dikompresi. Langkah selanjutnya adalah tampilan file oleh sistem, yang kemudian meminta konfirmasi dari pengguna untuk melanjutkan proses kompresi. Setelah proses kompresi selesai dieksekusi, hasilnya akan ditampilkan dan pengguna dapat menyimpan hasil tersebut.



Gambar 3.5. Activity Diagram Proses Dekompresi Algoritma Sequitur

Dalam proses Dekompresi algoritma Sequitur, Activity Diagram digunakan. File yang diterima sebagai input bukanlah file asli, melainkan file hasil kompresi sebelumnya menggunakan proses kompresi Sequitur. Pengguna akan menjalankan proses Dekompresi untuk mengembalikan data dari file tersebut menjadi data teks semula.

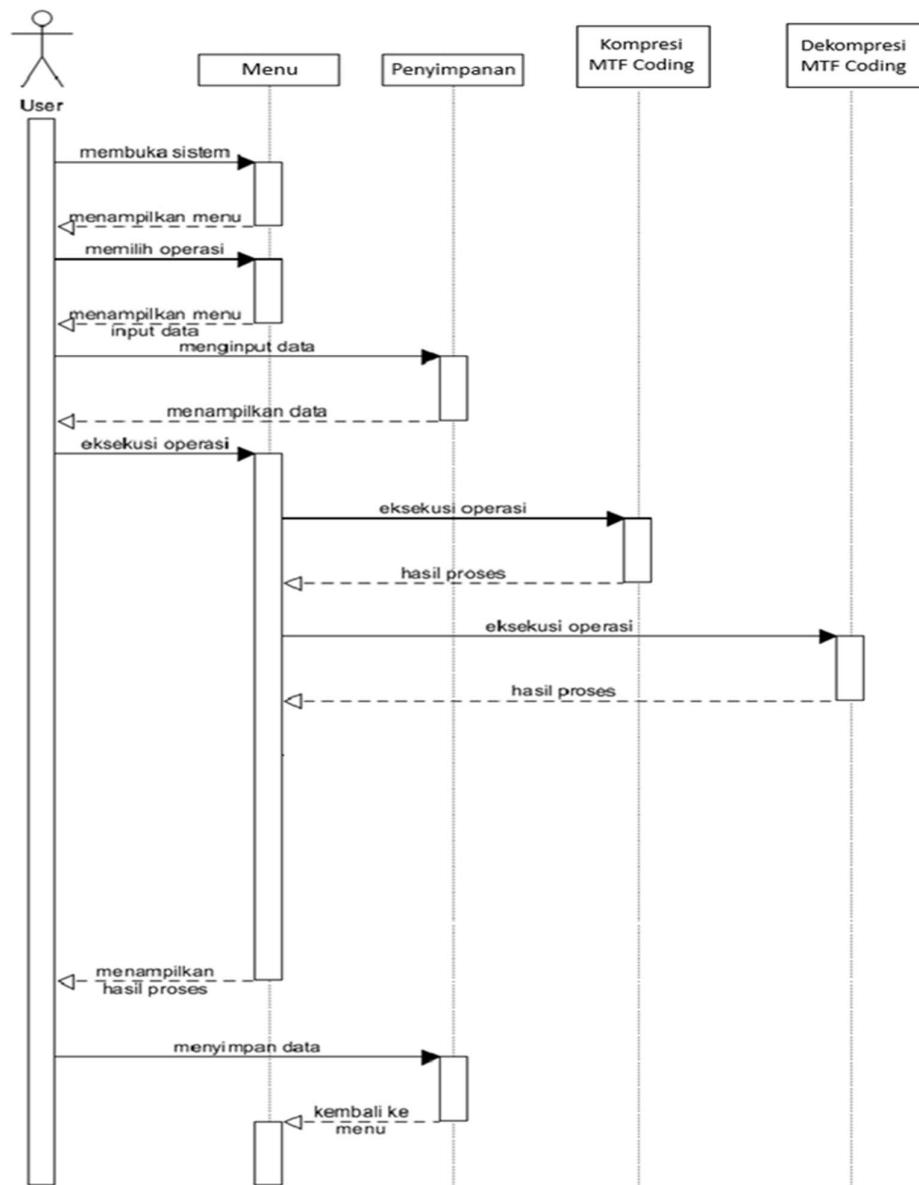


Gambar 3.6. Activity Diagram Proses Dekompresi Algoritma MTF Coding

Dalam Activity Diagram proses Dekompresi menggunakan algoritma MTF Coding, data hasil kompresi yang telah diproses sebelumnya dengan algoritma tersebut diinput. Setelah itu, sistem akan menampilkan data tersebut dan meminta pengguna untuk menekan tombol eksekusi guna menjalankan proses Dekompresi, mengembalikan data menjadi file teks asli. Hasil Dekompresi kemudian akan ditampilkan, memberikan opsi kepada pengguna untuk menyimpan data kembali.

3.2.3. Diagram Sequence

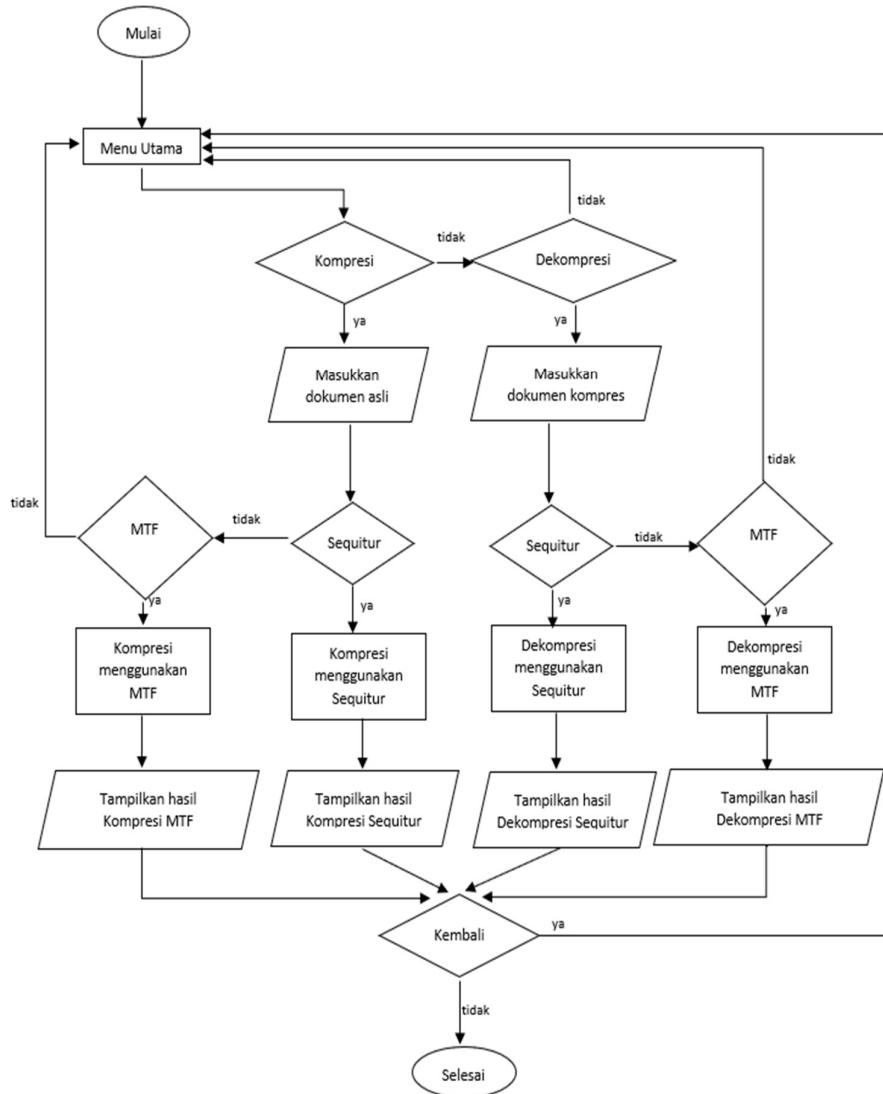
Diagram Sequence digunakan untuk mengilustrasikan interaksi antara objek dalam suatu urutan waktu.



Gambar 3.7. Diagram Sequence

3.3. Flow Chart

Diagram alur atau flowchart adalah representasi grafis yang menggambarkan urutan serta langkah-langkah proses yang terjadi dalam suatu sistem secara sistematis dan mudah dipahami. Diagram ini mencakup elemen-elemen seperti input, output, langkah-langkah proses, dan keputusan, yang diilustrasikan dengan bentuk-bentuk yang jelas dan dapat dimengerti.



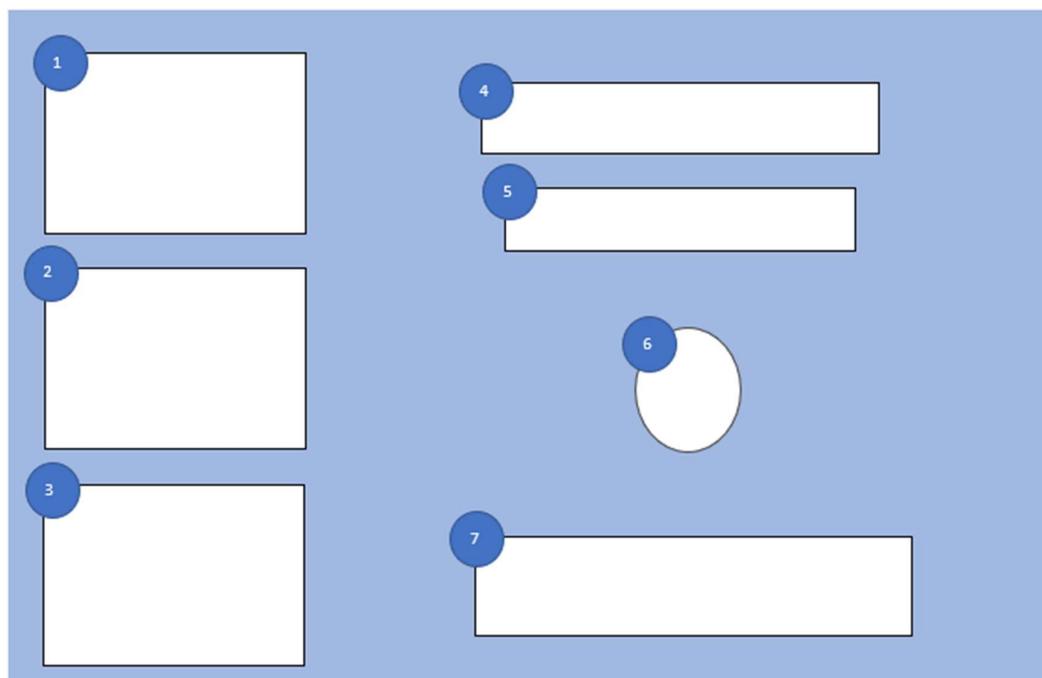
Gambar 3.8. Tampilan dari Flowchart

3.4. Tampilan Sistem

Tampilan Sitem membahas rancangan system yang akan dibuat, termasuk halaman-halaman utama dan fungsi dari setiap fiturnya. Terdapat 3 halaman dalam system ini: Halaman Utama dengan judul, halaman kompresi untuk fungsi kompresi, halaman dekompresi untuk fungsi dekompresi.

3.4.1. Tampilan menu Utama

Pada halaman Utama, yang merupakan halaman pertama yang muncul saat membuka siste, judul penelitian yang mendasari system yang ditampilkan



Gambar 3.9. Halaman Utama

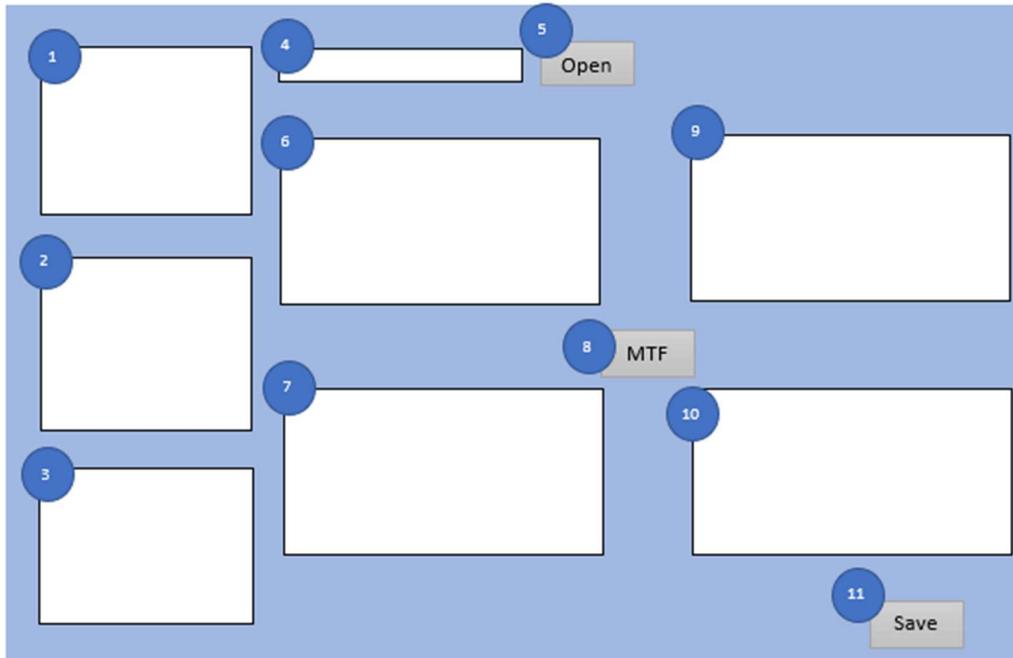
Keterangan pada tampilan menu utama tersebut dapat dilihat dari penjelasan pada Tabel 3.1:

Tabel 3.1. Keterangan Gambar Rancangan Halaman Menu Utama

No	Tipe	Keterangan
1	Button	Untuk menampilkan judul halaman yaitu halaman utama
2	Button	Untuk menampilkan kompresi
3	Button	Untuk menampilkan Dekompresi
4	Label	Untuk menampilkan judul penelitian
5	Label	Untuk menampilkan Nama dan NIM
6	Picture	Untuk menampilkan logo universitas
7	Label	Untuk Menampilkan Program studi, fakultas, nama universitas, dan tahun penelitian.

3.4.2. Tampilan menu Kompresi

Halaman Kompresi menampilkan fungsi-fungsi terkait kompresi dalam system, ini mencakup kemampuan untuk mengunggah file, menampilkan konten file, menjalankan proses kompresi, dan menampilkan hasil kompresi file.

**Gambar 3.10.** Halaman Kompresi

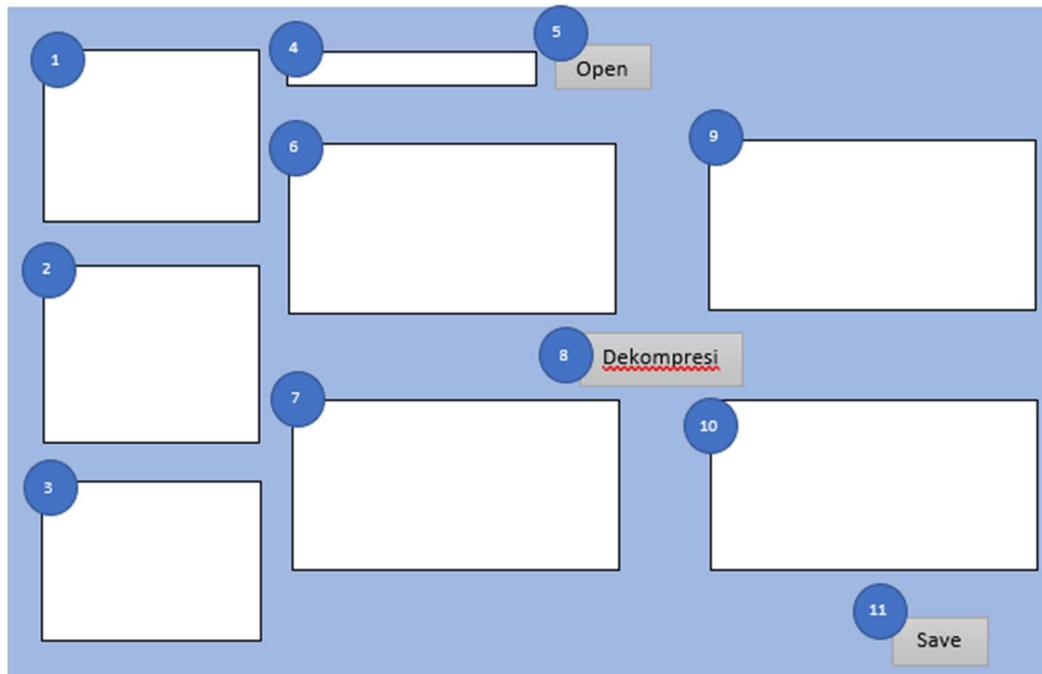
Informasi mengenai opsi yang tersedia dalam Tampilan Menu kompresi tersebut dapat dengan mudah ditemukan melalui deskripsi yang disediakan dalam table 3.2:

Tabel 3.2 Keterangan Gambar Rancangan Halaman kompresi

No	Tipe	Keterangan
1	Button	Untuk menampilkan judul halaman yaitu halaman utama
2	Button	Untuk menampilkan kompresi
3	Button	Untuk menampilkan Dekompresi
4	Textbox	Berisi informasi lokasi penyimpanan data yang akan diproses
5	Button	Untuk memproses atau mencari data yang akan diproses
6	Rich Textbox	Untuk menampilkan isi file asli yang akan di kompresi
7	Rich Textbox	Untuk menampilkan isi byte data dari file yang akan di kompresi
8	Button	Untuk kompresi MTF Coding
9	Rich Textbox	Untuk menampilkan byte data dari hasil kompresi <i>MTF Coding</i>
10	Rich Textbox	Untuk menampilkan hasil pengukuran parameter kompresi data
11	Button	Untuk menyimpan hasil kompresi

3.4.3. Tampilan menu Dekompresi

Di Tampilan Menu Dekompresi, tersedia fungsi untuk melakukan dekompresi data yang sebelumnya telah melalui proses kompresi. Terdapat opsi untuk memasukkan data, menampilkan data yang telah dikompresi, serta menjalankan proses dekompresi.

**Gambar 3.11.** Halaman Dekompresi

Untuk penjelasan mengenai komponen yang digunakan di atas ada pada tabel 3.3:

Tabel 3.3 Keterangan Gambar Rancangan Halaman Dekompresi

No	Tipe	Keterangan
1	<i>Button</i>	Untuk menampilkan judul halaman yaitu halaman utama
2	<i>Button</i>	Untuk menampilkan kompresi
3	<i>Button</i>	Untuk menampilkan Dekompresi
4	<i>Textbox</i>	Berisi informasi lokasi penyimpanan data yang akan diproses
5	<i>Button</i>	Untuk memproses atau mencari data yang akan di dekompresi
6	<i>Rich Textbox</i>	Untuk menampilkan isi <i>file</i> kompresi
7	<i>Rich Textbox</i>	Untuk menampilkan hasil pengukuran parameter dekompresi data
8	<i>Button</i>	Untuk melakukan dekompresi file
9	<i>Rich Textbox</i>	Untuk menampilkan isi byte dari hasil dekompresi
10	<i>Rich Textbox</i>	Untuk menampilkan teks asli setelah di dekompresi
11	<i>Button</i>	Untuk menyimpan hasil dekompresi

BAB 4

IMPLEMENTASI DAN PENGUJIAN SISTEM

4.1. Implementasi Sistem

Setelah menyelesaikan tahap analisis dan perancangan sistem, langkah berikutnya adalah mengimplementasikan sistem tersebut. Implementasi sistem merupakan proses mengubah rancangan yang telah disusun menjadi sebuah sistem yang siap dioperasikan.

Rancangan ini kemudian diwujudkan dalam bentuk aplikasi dengan bantuan Integrated Development Environment (IDE), sebuah perangkat lunak yang diciptakan untuk mendukung pengembangan, pemrograman, dan pengujian perangkat lunak menggunakan bahasa pemrograman tertentu.

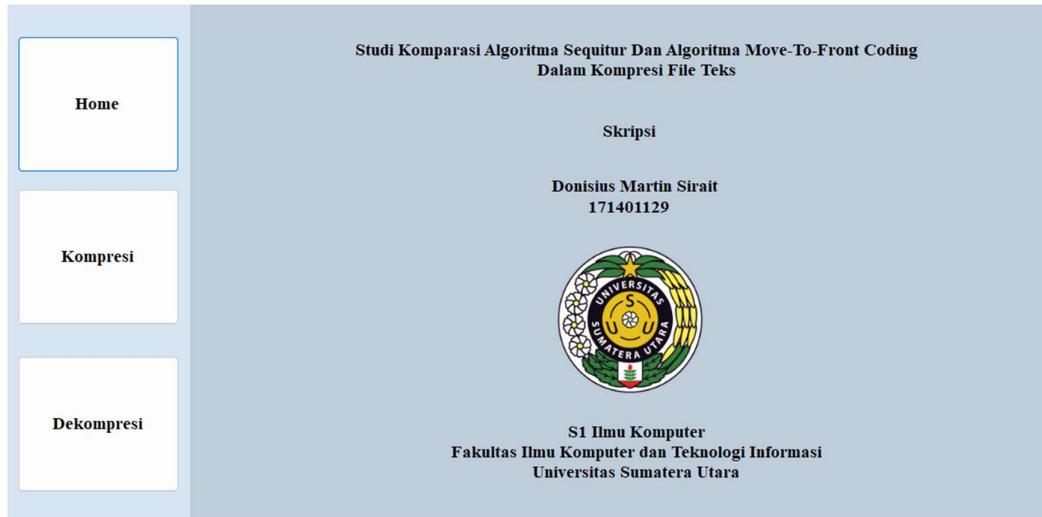
Pada tahap implementasi ini, rancangan sistem yang telah disusun diubah menjadi sebuah program aplikasi desktop yang berjalan di Windows dengan menggunakan bahasa pemrograman C#. Proses ini dilakukan menggunakan perangkat lunak IDE Sharp Develop 5.1.

Aplikasi yang dihasilkan memiliki tiga halaman yang dimana dengan fungsi yang berbeda: halaman home, halaman kompresi, halaman dekompresi.

4.1.1. Halaman Home

Halaman home adalah tampilan yang akan pertama kali ditampilkan saat pertama kali sistem dijalankan. Pada halaman ini, ditampilkan informasi tentang penelitian, termasuk judul penelitian, nama dan NIM penulis, logo universitas, nama program studi, nama fakultas, dan nama universitas.

Tampilan halaman home dalam program dapat dilihat pada Gambar 4.1.

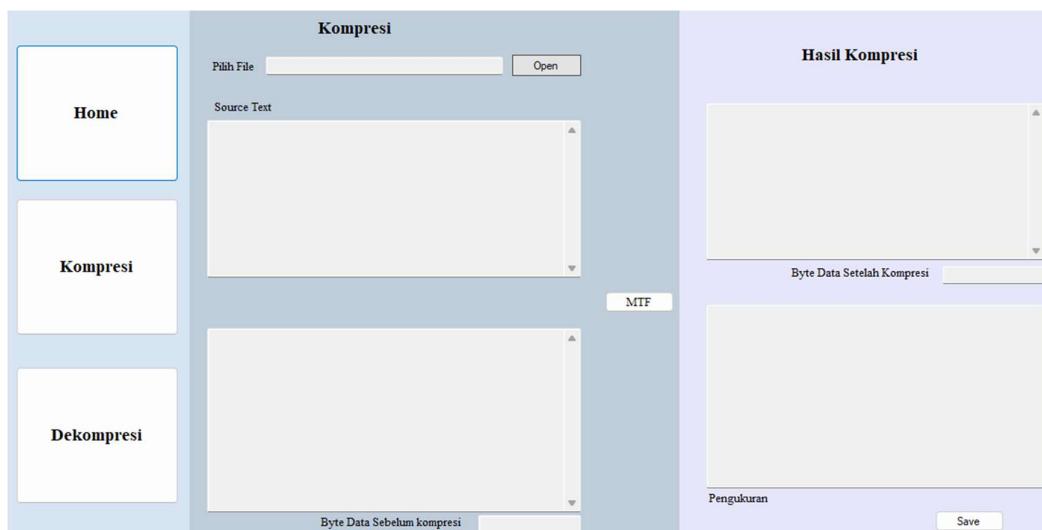


Gambar 4.1. Halaman Home

4.1.2. Halaman Kompresi

Halaman kompresi adalah halaman yang menyediakan menu untuk proses kompresi data. Halaman ini dilengkapi dengan beberapa elemen seperti textbox, label, dan tombol yang masing-masing memiliki fungsi tertentu. Terdapat tombol untuk memilih file yang akan dikompresi dari direktori komputer pengguna, tombol untuk mengeksekusi kompresi data, dan tombol untuk menyimpan hasil kompresi. Selain itu, terdapat textbox yang menampilkan data asli dan data hasil kompresi dalam format string biner, serta kotak teks yang menunjukkan ukuran data dan hasil perhitungan kompresi file tersebut.

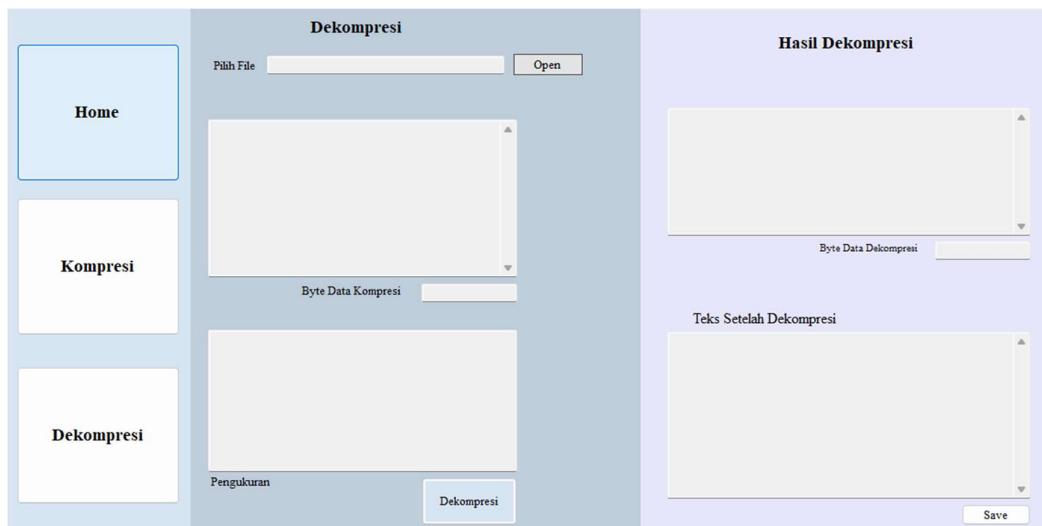
Tampilan halaman kompresi ini dapat dilihat pada Gambar 4.2.



Gambar 4.2. Halaman Kompresi

4.1.3. Tampilan Dekompresi

Halaman dekompresi adalah halaman yang menyediakan menu untuk proses dekompresi. Tampilan dari halaman dekompresi terlihat mirip dengan halaman kompresi. Pada halaman dekompresi terdapat tombol untuk memilih file dari direktori komputer yang sebelumnya telah di kompresi dan di simpan, tombol untuk menjalankan proses dekompresi, textbox yang akan menampilkan data sebelum dan sesudah dekompresi, textbox untuk pengukuran data, dan tombol save untuk menyimpan hasil dekompresi.



Gambar 4.3. Halaman Dekompresi

4.2. Pengujian

Bahan uji adalah sekumpulan data yang digunakan untuk menilai kinerja algoritma yang sedang diteliti. Dalam hal ini kompresi algoritma Sequitur dan MTF Coding. Kumpulan data ini biasanya mencakup berbagai jenis file atau informasi dengan karakteristik berbeda, seperti teks, gambar, vidio, suara, dan lain-lain.

Pada penelitian ini, bahan uji yang digunakan adalah data berbentuk file teks. File teks adalah dokumen atau informasi tertentu yang disimpan dalam bentuk teks saja. Dalam penelitian ini, penulis menggunakan file teks dengan format .txt.

File teks berformat .txt adalah data yang berisi teks biasa tanpa format khusus, hanya menyimpan teks tanpa format font, warna, atau format lainnya. Jenis file ini dapat dibuka dengan berbagai program pengolah teks.

Dalam pengujian, dikarenakan keterbatasan penulis. kompresi yang dilakukan dengan sistem adalah algoritma MTF Coding. Kompresi algoritma Sequitur akan dilakukan secara manual.

4.1.4. Pengujian Proses Kompresi

Ada beberapa dataset teks yang dipergunakan untuk menguji sistem ini. Penulis akan menyediakan beberapa teks yang akan dipilih dalam pengujian sistem.

Untuk melihat proses pengujian kompresi dalam sistem, dapat dilihat pada tabel 4.1.

Tabel 4.1. Daftar Bahan Uji

No.	Nama File	Ukuran (byte)	Keterangan
1	a.txt	1	Mengandung satu karakter huruf "a" dari The Artificial Corpus.
2	abjad.txt	26	Berisi daftar huruf abjad.
3	abcabcabc.txt	9	Berisi huruf abcabcabc.
4	nama.txt	22	Berisi nama penulis.

5	teks huruf acak random.txt	23	Berisi teks huruf acak random.
6	π (pi).txt	50	Berisi 50 digit pertama dari π .

4.3. Proses Pengujian Sistem

Pengujian sistem adalah langkah untuk mengidentifikasi hasil dari penerapan sistem yang telah dirancang sebelumnya. Tahapan ini dilakukan pada file teks yang telah ditentukan sebagai data uji. Proses pengujian sistem dibagi menjadi dua bagian utama, yaitu pengujian kompresi dan pengujian dekompresi.

Pengujian kompresi bertujuan untuk memastikan bahwa sistem dapat berjalan dengan baik dan mampu mengompresi data uji sesuai dengan yang diharapkan. Sedangkan pengujian dekompresi bertujuan untuk memastikan bahwa data yang telah dikompresi dapat dikembalikan ke bentuk aslinya tanpa kehilangan informasi setelah melalui proses dekompresi.

4.1.5. Proses Pengujian Kompresi Data Algoritma MTF Coding

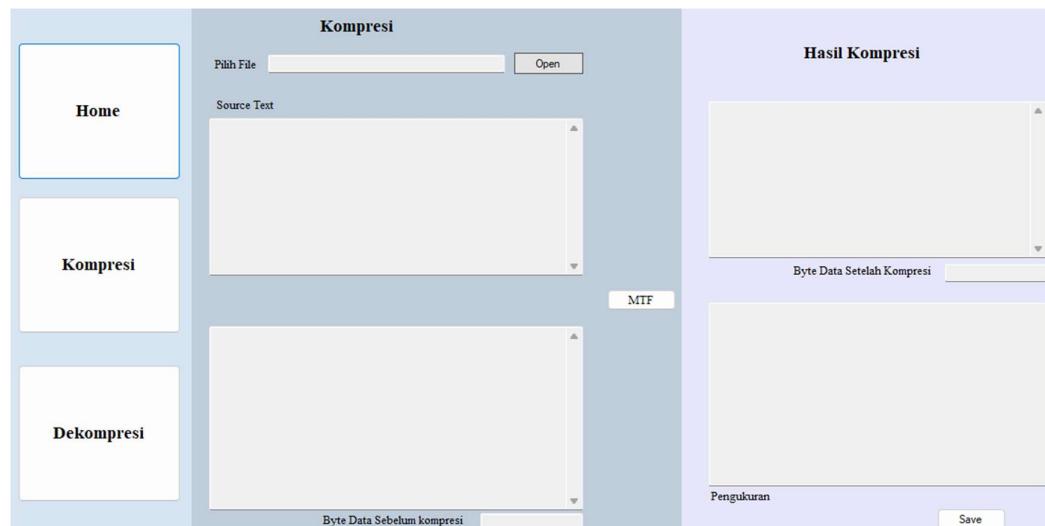
Pada proses pengujian kompresi data akan ada dua sistem pengujian yaitu Sequitur akan dikompresi secara manual dikarenakan keterbatasan oleh penulis, dan kompresso MTF Coding akan dikompresi menggunakan program yang telah dirancang sebelumnya.

Proses pengujian kompresi MTF Coding dimulai dengan menjalankan program atau sistem yang telah diimplementasikan sebelumnya. Setelah program terbuka, halaman home akan menampilkan informasi seperti judul penelitian, logo universitas, nama penulis, jurusan, fakultas, dan nama universitas.

Langkah selanjutnya adalah dengan menekan tombol menu kompresi di sebelah kiri untuk masuk ke halaman kompresi, dan proses kompresi akan dilakukan. Setelah memilih menu kompresi, maka sistem akan menampilkan antarmuka kompresi.

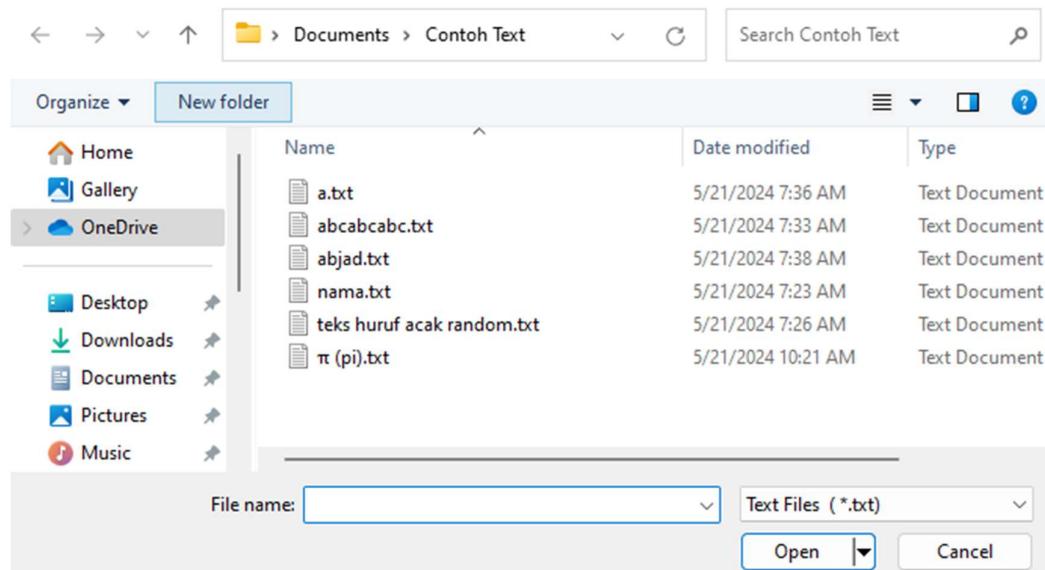
Berikut adalah langkah-langkah untuk menjalankan proses kompresi:

1. Klik tombol *Open* untuk memilih file teks, seperti yang ditunjukkan dalam Gambar 4.4.



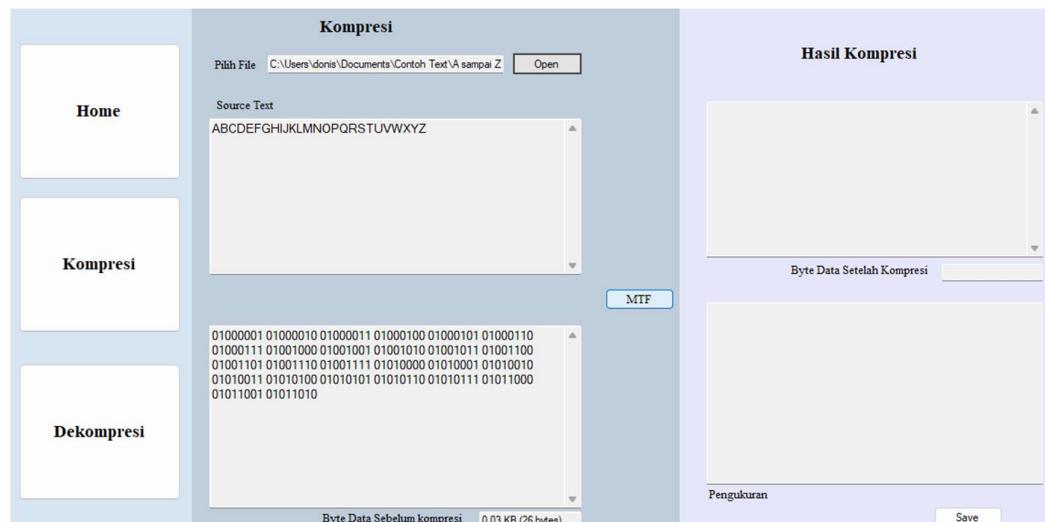
Gambar 4.4. Tombol open untuk memilih file teks yang akan dikompresi.

2. Pilih file teks yang ingin dikompresi, sesuai petunjuk yang tertera pada Gambar 4.5.



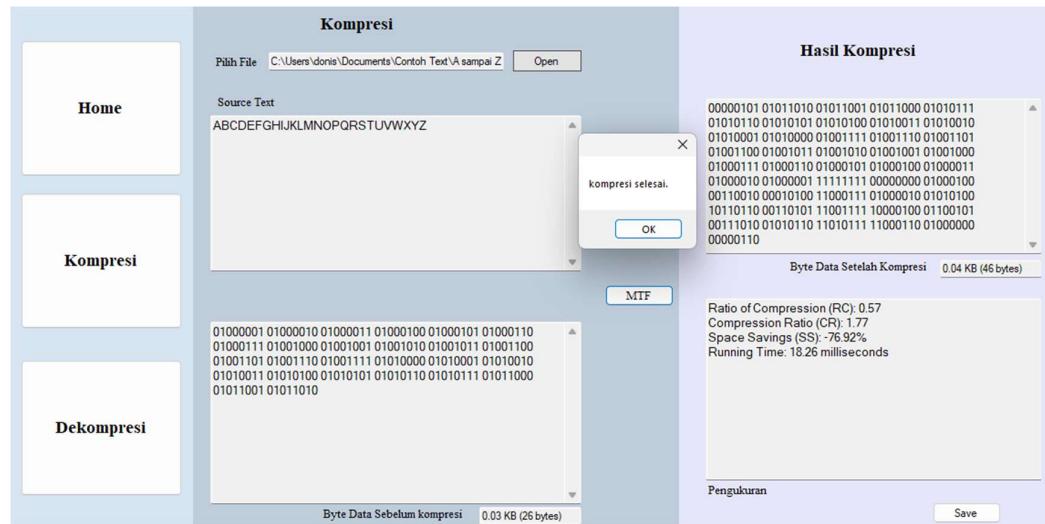
Gambar 4.5. Open File Dialog

3. Setelah memilih file teks, sistem akan menampilkan isi dari file teks serta hasil konversi ke kode biner dan menampilkan byte dari file tersebut. Selanjutnya untuk melakukan kompresi, pengguna menekan tombol MTF untuk menampilkan hasil dari kompresi menggunakan algoritma MTF Coding. Seperti yang dinunjukkan pada Gambar 4.6.



Gambar 4.6. Tampilan dari isi file teks

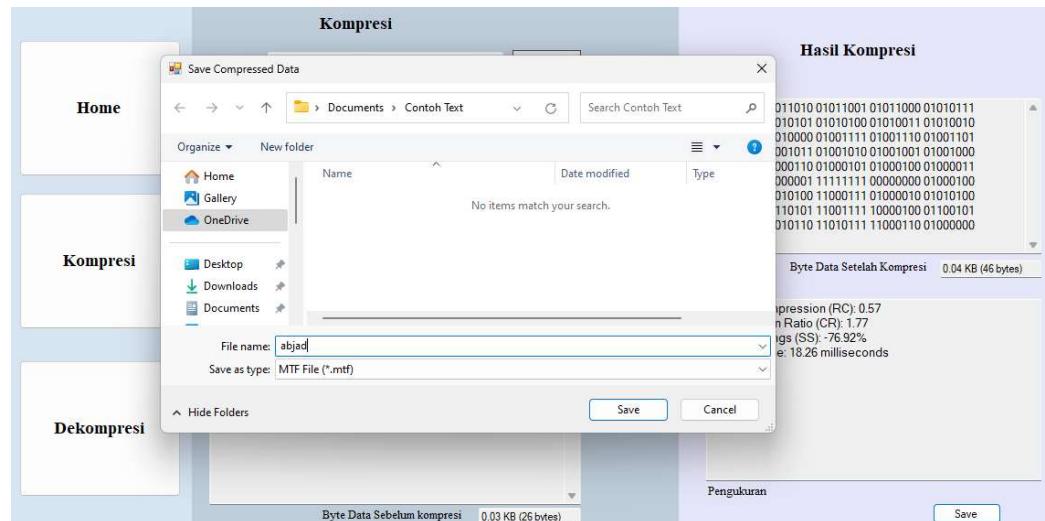
4. Setelah pengguna menekan tombol MTF, sistem akan memulai proses kompresi menggunakan kompresi MTF Coding, Setelah proses selesai, sistem akan menampilkan perhitungan dari *Compression Ratio*, *Ratio of Compression*, *Space Saving*, dan *Running Time*-nya seperti pada Gambar 4.7.



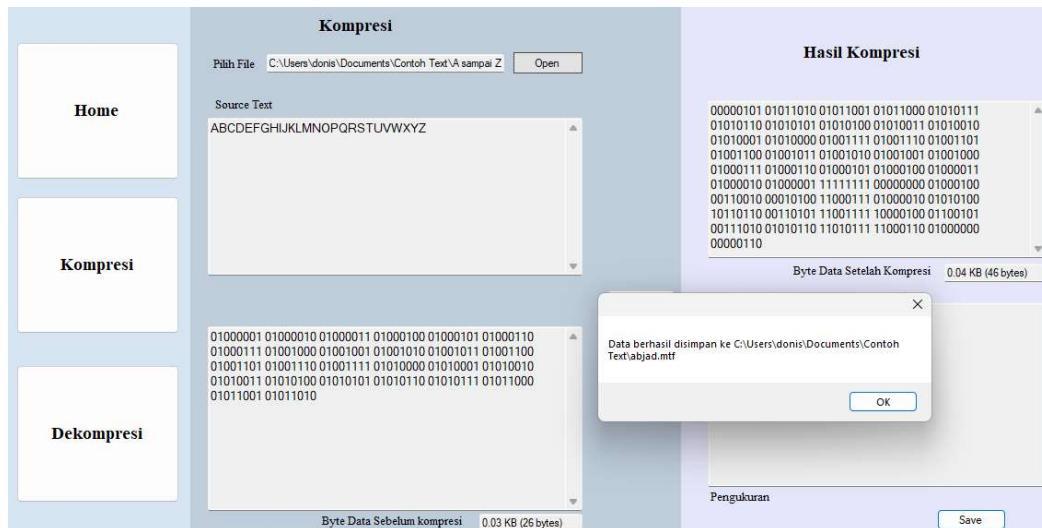
Gambar 4.7. Tampilan hasil dari kompresi algoritma MTF Coding

5. Pengguna dapat menyimpan hasil dari kompresi dengan menekan tombol *Save*.

Sistem akan menyimpan data dengan ekstensi .mtf seperti pada gambar 4.8. dan 4.9.



Gambar 4.8. Save file dialog untuk menyimpan data algoritma MTF Coding



Gambar 4.9. Kompresi berhasil disimpan

4.1.6. Proses Pengujian Kompresi Data Algoritma Sequitur

Proses pengujian kompresi data dengan menggunakan algoritma Sequitur dilakukan secara manual dikarenakan keterbatasan penulis. Langkah-langkah pengujian kompresi adalah sebagai berikut:

1. Memilih salah satu data dari daftar bahan uji, misalnya: abcabcabc.txt.
 2. Melakukan perhitungan seperti di bawah ini:

Contoh Perhitungan Sequitur

String : "abcabcabc"

Seleksi teks : ab, bc, ca, ab, bc, ca, ab, bc

Perulangan terjadi pada ‘SI’, maka diganti dengan simbol non-terminal ‘A’ \rightarrow ‘ab’.

Perubahan string : “AcAcAc”

dilakukan seleksi teks kembali : Ac, cA, Ac, cA, Ac

Perulangan terjadi pada Ac, maka diganti dengan ‘B’ → ‘Ac’

Dilakukan seleksi teks kembali : BBB

Karena simbol A digunakan sekali dan tergantikan pada simbol $B \rightarrow Ac$, maka simbol A dihapus dan dimasukkan ke $B \rightarrow abc$.

Sehingga rule yang dihasilkan hanya B.

Maka, hasil kompresi : "BBB"

Contoh dalam bentuk tabel:

Tabel 4.2. Contoh proses kompresi sequitur

String	Simbol Berulang	Rule	Perubahan String	Simbol yang Dihapus
abcabcabc	ab	$A \rightarrow ab$	AcAcAc	-
AcAcAc	Ac	$A \rightarrow ab$ $B \rightarrow Ac$	BBB	A
BBB	abc	$B \rightarrow abc$	BBB	-

Untuk mengetahui ukuran teks yang akan dikompresi, terlebih dahulu diurutkan berdasarkan frekuensi kemunculan karakter terbanyak hingga terkecil, dan menghitung bit dari setiap karakternya. Berikut adalah perhitungan dalam bentuk tabel:

Tabel 4.3. Contoh Proses Perhitungan Sebelum Kompresi

KARAKTER	FREKUENSI	DESIMAL	BINER	BIT	BIT X FREKUENSI
a	3	97	0110 0001	8	24
b	3	98	0110 0010	8	24
c	3	99	0110 0011	8	24
Total	9	-	-	-	72

Berikut setelah dilakukan kompresi “BBB”:

Tabel 4.4. Contoh Proses Perhitungan Kompresi Algoritma Sequitur

KARAKTER	FREKUENSI	DESIMAL	BINER	BIT	BIT X FREKUENSI
B	3	66	0100 0010	8	24
Total	3	-	-	-	24

Maka hasil dari pengukuran kompresi menggunakan kompresi algoritma Sequitur Sebagai berikut:

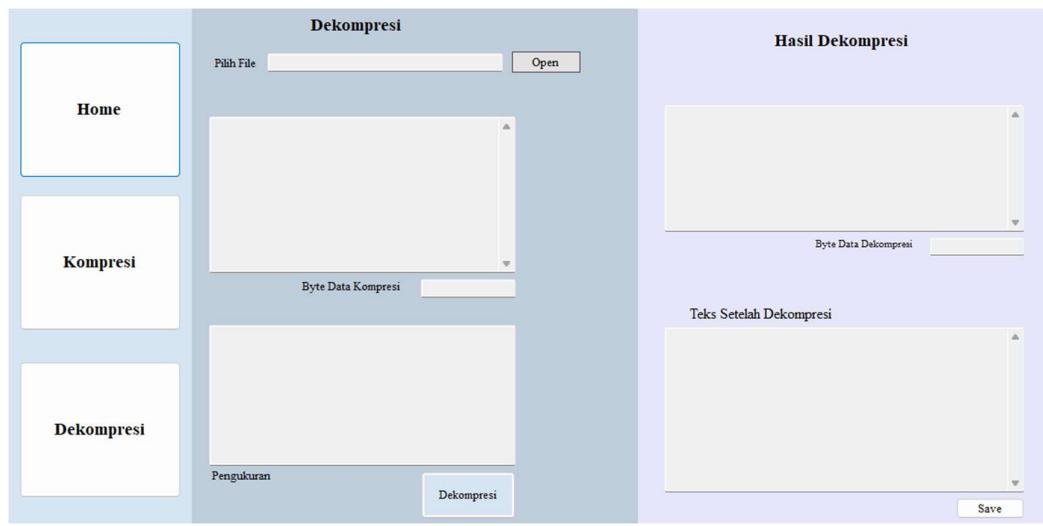
$$\text{Ratio of Compression (RC)} = 9 / 3 = 3$$

$$\text{Compression Ratio (CR)} = 3 / 9 = 0.3$$

$$\text{Space Savings (SS)} = 100\% - 30\% = 70\%$$

4.1.7. Proses Pengujian Dekompresi Data Algoritma MTF Coding

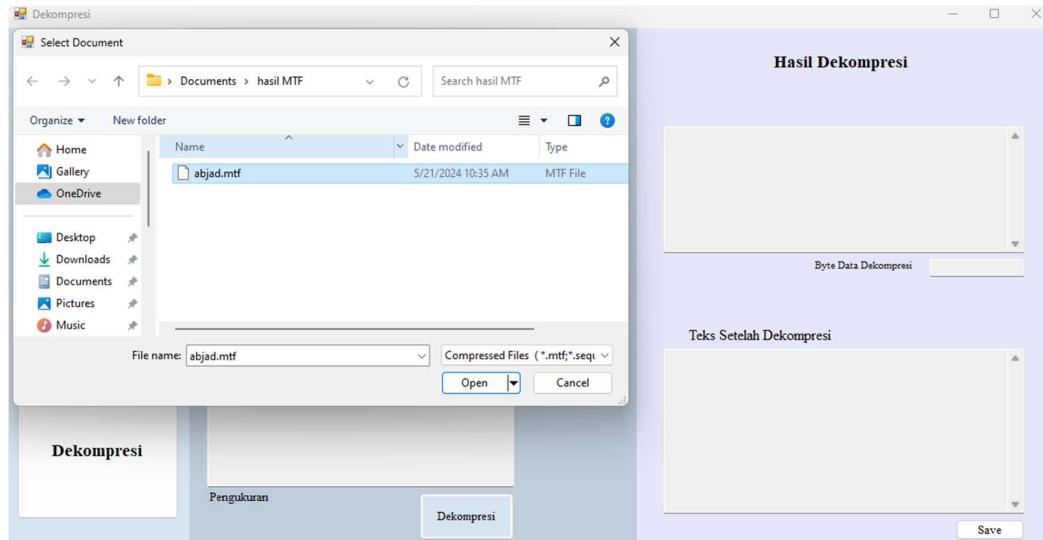
Seperti pada proses pengujian kompresi, pilih tombol menu dekompresi, maka tampilan halaman dekompresi akan terbuka seperti pada Gambar 4.10.



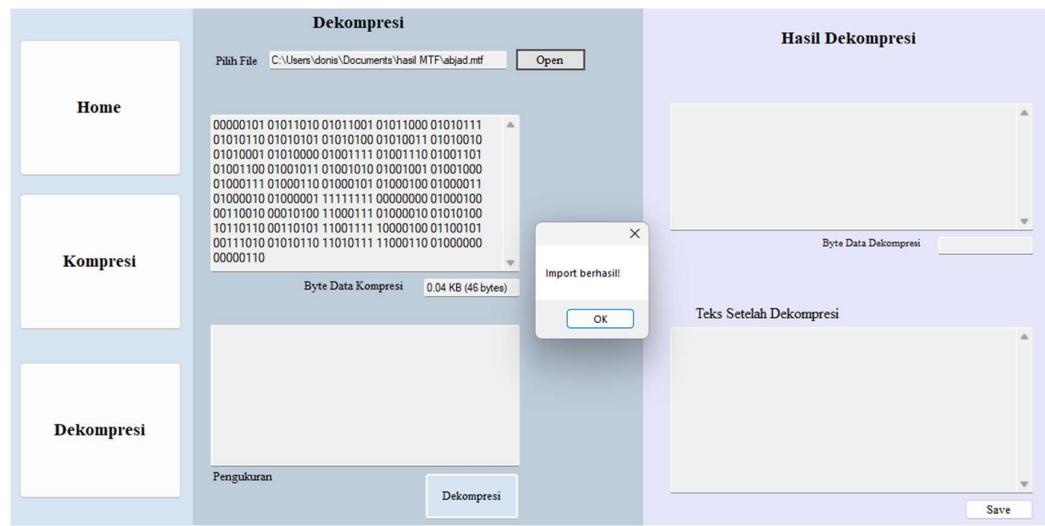
Gambar 4.10. Tampilan Halaman menu dekompreksi

Selanjutnya ikuti langkah berikut ini:

1. Tekan tombol open untuk memilih file yang telah di kompresi sebelumnya, lalu pilih file dengan format .mtf, dari direktori komputer, lalu klik open. Sistem akan membuka data yang dipilih, seperti pada Gambar 4.11. dan Gambar 4.12.

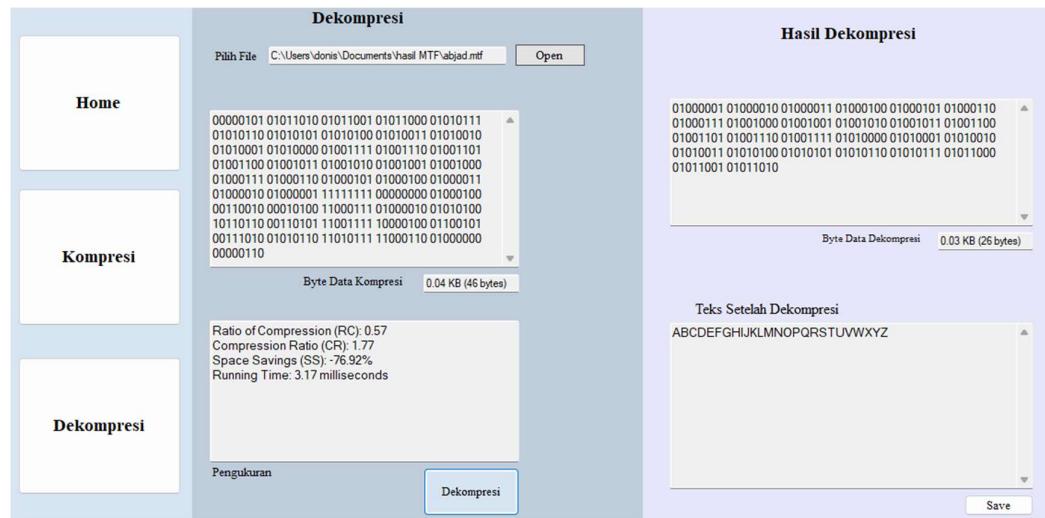


Gambar 4.11. Memilih file yang akan didekompreksi



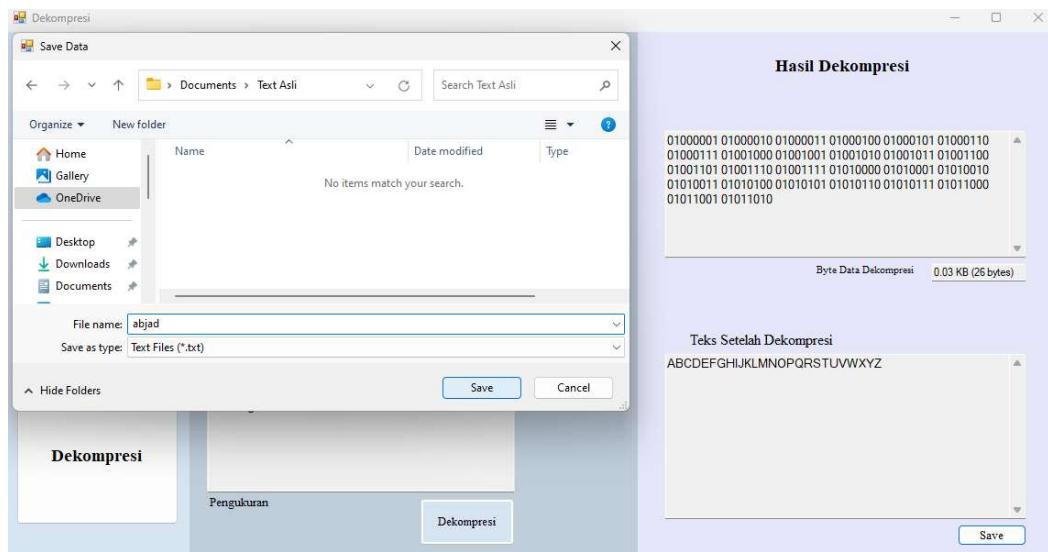
Gambar 4.12. Tampilan dari data yang akan didekompresi

2. Tekan tombol dekompresi untuk mengembalikan file yang telah dikompresi dengan format .mtf sebelumnya. Setelah dilakukan dekompresi, sistem akan menampilkan byte data dekompreksi/ byte data asli, dan juga akan menampilkan teks kembali ke ukuran aslinya.

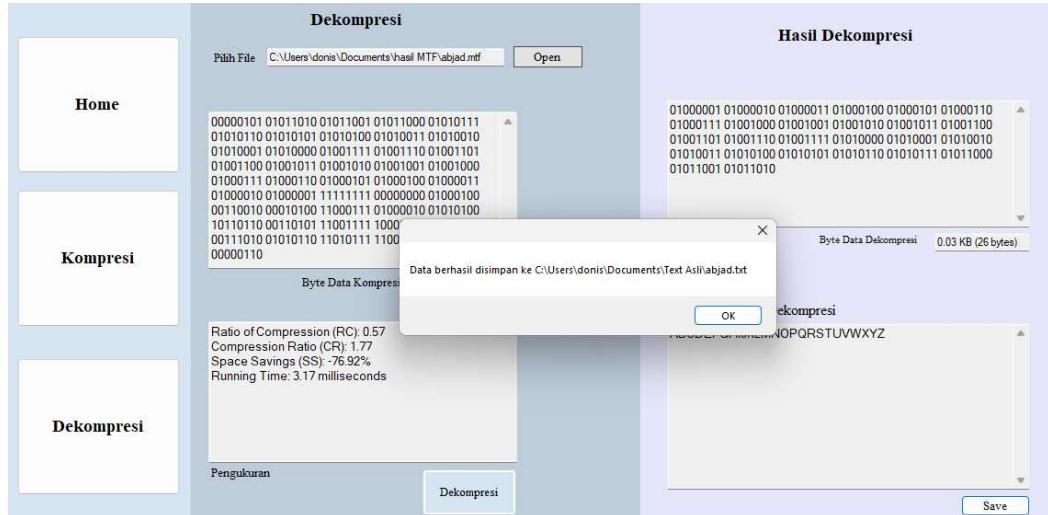


Gambar 4.13. Proses Dekompreksi

3. Pengguna dapat menyimpan hasil dari dekompreksi dengan menekan tombol *Save*. Sistem akan menyimpan data dalam bentuk format .txt seperti pada Gambar 4.14. dan Gambar 4.15.



Gambar 4.14.Masukkan nama file yang akan disimpan



Gambar 4.15. Hasil Dekompresi Berhasi Disimpan

4.1.8. Proses Pengujian Dekompresi Data Algoritma Sequitur

Proses pengujian dekompresi data dengan menggunakan algoritma Sequitur dilakukan secara manual dikarenakan keterbatasan penulis. Langkah-langkah pengujian dekompresi adalah sebagai berikut:

1. Melakukan dekompresi dari file yang sebelumnya telah dikompresi.

2. Pada saat kompresi dilakukan, kita telah menyimpan rule. Untuk melakukan dekompresi, rule yang disimpan sebelumnya akan digunakan kembali untuk mengembalikan string ke dalam bentuk semula, seperti contoh pada file abjad.txt pada tabel berikut:

Tabel 4 5. Tabel Dekompresi Sequitur

String	Rule	Keterangan	Perubahan String
BBB	$B \rightarrow Ac$	Simbol B dikembalikan ke string awal, yaitu ‘Ac’.	AcAcAc
AcAcAc	$A \rightarrow ab$	Simbol A dikembalikan ke string awal, yaitu ‘ab’	abcabcabc

Pada saat proses dekompresi harus dilakukan secara berurutan dari rule terakhir, seperti pada contoh, rule terakhir adalah $B \rightarrow Ac$, maka rule B yang harus digunakan terlebih dahulu, begitu juga seterusnya sampai string kembali ke semula.

4.4. Hasil Pengujian

Hasil dari pengujian sistem merujuk pada informasi atau hasil yang diperoleh selama proses pengujian. Pada penelitian ini, evaluasi dilakukan terhadap kinerja algoritma kompresi yang terintegrasi dalam sistem terhadap sejumlah data yang telah diuji sebelumnya.

4.1.9. Hasil Dari Pengujian Kompresi Pada data Bahan Uji

Hasil dari pengujian sistem dalam penelitian ini melibatkan evaluasi fungsi dari sistem dalam melakukan kompresi dan dekompresi, serta kinerja sistem dalam proses kompresi file data dalam format .txt. Informasi dari hasil pengujian dapat dilihat dalam Tabel 4.6.

Tabel 4.6. Hasil Pengujian Kompresi

no	nama	Ukuran Awal	Algoritma	Ukuran Hasil	RC	CR	SS	RT (ms)
1	a.txt	1	Sequitur	1	1	1	0.00%	-
			MTF Coding	5	0.2	5	-400%	0.04
2	abjad.txt	26	Sequitur	1	1	1	0.00%	-
			MTF Coding	46	0.57	1.77	-76.92%	0.08
3	abcabcabc.txt	9	Sequitur	3	3	0.3	70.00%	-
			MTF Coding	9	1	1	0.00%	0.03
4	nama.txt	22	Sequitur	20	1.1	0.91	9.00%	-
			MTF Coding	25	0.88	1.14	-13.64%	0.04
5	random.txt	23	Sequitur	18	1.27	0.78	22.00%	-
			MTF Coding	30	0.77	1.3	-30.43%	0.08
6	π (pi).txt	50	Sequitur	41	1.22	0.82	18.00%	-
			MTF Coding	39	1.28	0.78	22.00%	0.08

4.1.10. Hasil Dari Pengujian Dekompresi Pada data Bahan Uji

Tabel 4.7. Hasil Pengujian Dekompresi

no	nama	Algoritma	Ukuran Sebelum (bytes)	Ukuran Sesudah (bytes)	Running Time	Ukuran Asli
1	a.txt	Sequitur	1	1	-	1
		MTF Coding	5	1	0.04	
2	abjad.txt	Sequitur	26	26	-	26
		MTF Coding	46	26	0.08	
3	abcabcabc.txt	Sequitur	3	9	-	9
		MTF Coding	9	9	0.03	
4	nama.txt	Sequitur	20	22	-	22

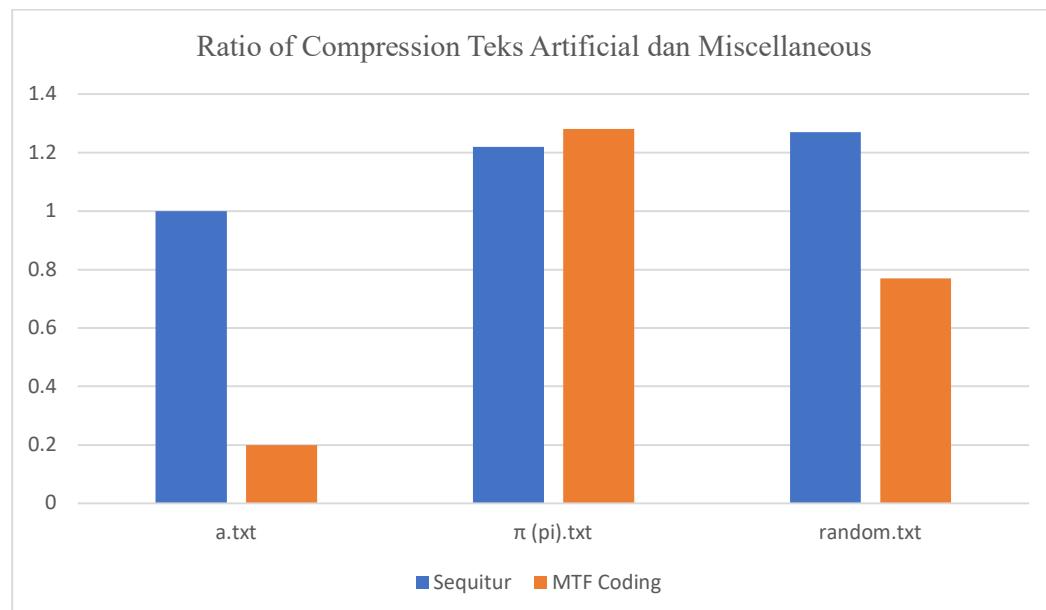
		MTF Coding	25	22	0.04	
5	random.txt	Sequitur	18	23	-	23
		MTF Coding	30	23	0.08	
6	π (pi).txt	Sequitur	41	50	-	50
		MTF Coding	39	50	0.04	

4.1.11. Hasil pengujian dalam bentuk Diagram

Untuk memberikan gambaran yang lebih jelas mengenai proses kompresi dan dekompresi yang telah di sajikan dalam Tabel 4.6. dan Tabel 4.7., Hasil-hasil tersebut dirangkum dalam bentuk grafik diagram yang dikelompokkan berdasarkan jenis data. Aspek yang dibandingkan adalah *Ratio of Compression* pada kompresi dan dekompresi.

1. Artificial Corpus dan Miscellaneous Corpus

Hasil Perbandingan *Ratio of Compression* dari *Artificial Corpus* dan *Miscellaneous Corpus* dapat dilihat pada Gambar 4.16.

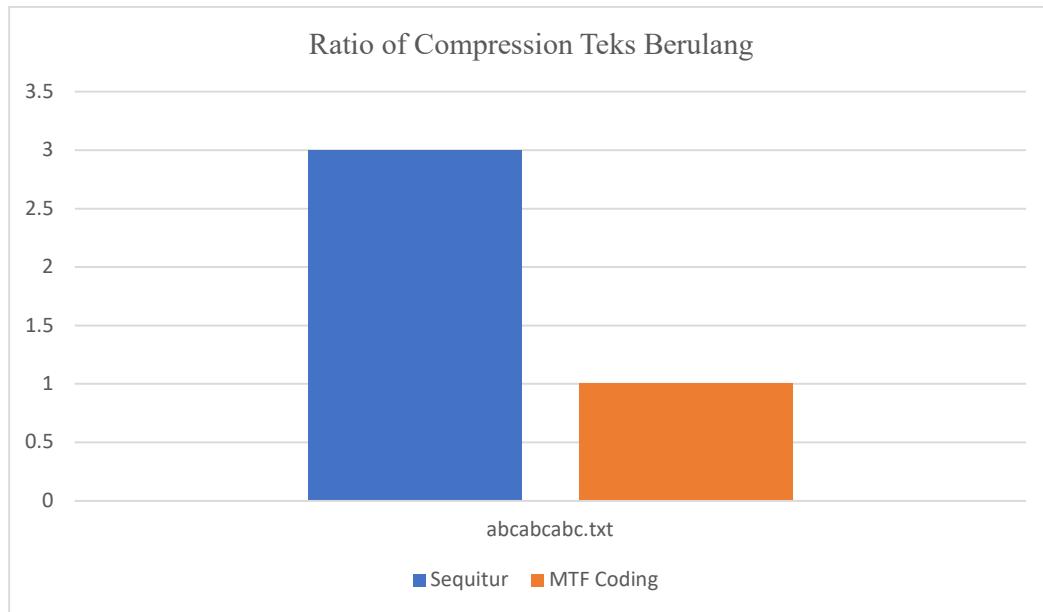


Gambar 4 16. Perbandingan Ratio of Compression dari Artificial Corpus dan Miscellaneous Corpus.

Pada Gambar 4.16. menunjukkan MTF Coding tidak dapat melakukan kompresi pada file a.txt, dan pada file π (pi).txt MTF Coding lebih efektif dalam kompresi dikarenakan banyaknya angka, dikarenakan MTF Coding akan lebih efisien dengan teks yang menggunakan banyak angka, dan pada file random.txt terlihat algoritma Sequitur lebih baik dalam mengkompresi data.

2. Hasil Perbandingan Dari Data Berulang

Penggunaan algoritma Sequitur menunjukkan hasil yang sangat baik saat mengompresi data berulang seperti abcabcabc.txt. Perbandingan *Ratio of Compretion* dapat dilihat pada Gambar 4.17.:

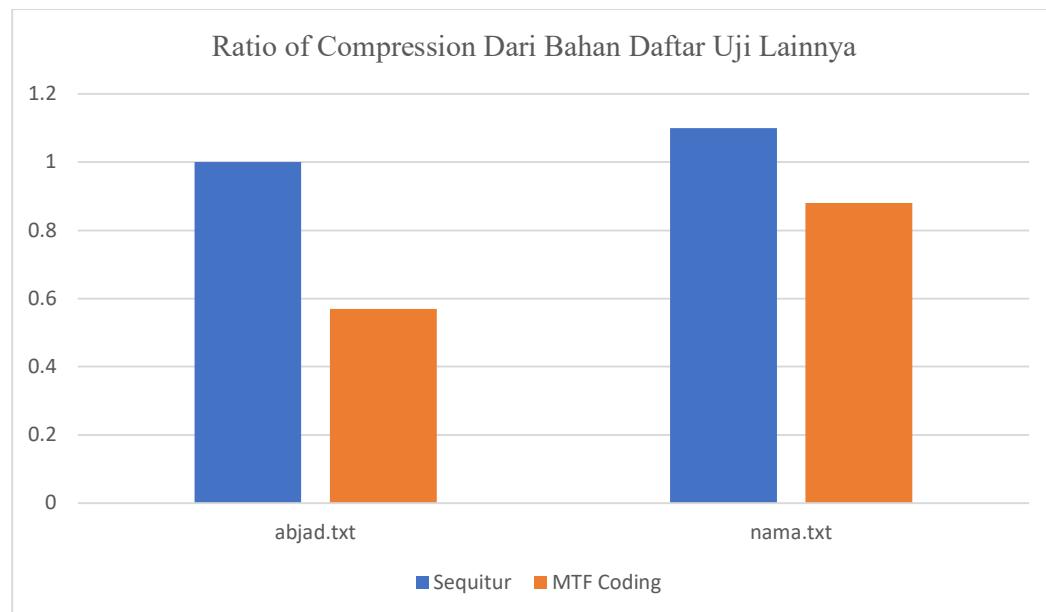


Gambar 4.17. Perbandingan Ratio of Compression Teks Berulang

Terlihat pada gambar 4.17. perbandingan untuk teks berulang algoritma Sequitur lebih baik dalam kompresi pada file teks berulang.

3. Hasil Perbandingan Dari Bahan Daftar Uji Lainnya.

Hasil dari kompresi menggunakan algoritma Sequitur dan MTF Coding terlihat bahwa algoritma Sequitur jauh lebih baik dan efektif dalam kompresi file teks dari MTF Coding.



Gambar 4.18. Perbandingan Ratio of Compression abjad.txt dan nama.txt

Hasil dari perbandingan kompresi algoritma Sequitur dan algoritma MTF Coding pada Gambar 4.18. terlihat algoritma Sequitur lebih efektif dalam kompresi pada file abjad.txt dan nama.txt.

Sedangkan untuk perbandingan kecepatan (*Running Time*), penulis tidak dapat menampiknya dikarenakan algoritma Sequitur yang dikerjakan secara manual.

4.4.4. Pengamatan Dari Hasil Pengujian

Berdasarkan hasil dari pengujian, baik itu secara sistem program atau manual yang telah ditampilkan sebelumnya dalam tabel kompresi dan dekompresi, beberapa temuan mengenai kinerja kompresi file teks dari algoritma Sequitur dan algoritma MTF Coding yang telah diuji dapat disimpulkan, antara lain:

1. Algoritma MTF Coding akan lebih efektif apabila banyak angka pada teks. Ini dikarenakan MTF coding menangani pola-pola ini dengan sangat efisien karena elemen yang sering diulang tetap berada di dekat awal daftar. Misalnya, dalam teks yang berisi angka, angka-angka tertentu mungkin sering muncul berulang kali dalam urutan yang berbeda. MTF coding dengan cepat menyesuaikan urutan elemen dalam daftar untuk mengoptimalkan pengodean elemen yang sering muncul.
2. Algoritma Sequitur sangat efektif untuk mengompresi data yang mengandung kata-kata atau pola yang berulang. Algoritma ini bekerja dengan mendeteksi dan menggantikan pola berulang dengan aturan produksi. Setiap kali sebuah pola terdeteksi lebih dari satu kali, pola tersebut digantikan dengan simbol baru yang mewakili pola tersebut. Proses ini sangat efisien untuk data dengan banyak pola berulang pada berbagai tingkat, membuat Sequitur sangat berguna dalam kompresi teks dan aplikasi lain yang melibatkan pola berulang.
3. Pada proses Kompresi dan dekompresi, *size* dari algoritma Sequitur akan selalu lebih kecil dari algoritma MTF Coding, apabila file teks yang akan di kompresi memiliki sedikit atau tidak ada angka. Itu dikarenakan MTF Coding lebih efektif dalam kompresi angka, sedangkan algoritma Sequitur akan lebih efektif pada pola berulang.
4. Kompresi pada *file* dari *The Miscellaneous Corpus* yaitu π (pi).txt. algoritma MTF Coding menunjukkan kinerja yang lebih baik. Hal ini disebabkan banyaknya angka yang berulang.

Dari hasil pengujian sistem terhadap proses kompresi dan dekompresi pada *file* data yang sebelumnya, kita dapat menarik beberapa informasi yang relevan. Infomasi tersebut memberikan gambaran tentang kinerja kedua algoritma dalam menangani kompresi data dengan berbagai karakteristik yang beragam.

BAB 5

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan hasil dari studi komparasi dan pengujian sistem dalam penelitian ini, terdapat beberapa kesimpulan yang berkaitan dengan kinerja algoritma Sequitur dan algoritma MTF Coding terhadap file teks. Beberapa kesimpulan yang dapat diambil antara lain:

1. Algoritma Sequitur tidak akan menambah jumlah bit karena hanya akan menggantikan pola berulang dengan simbol yang lebih singkat, membuat aturan produksi yang efisien, dan mengurangi pengulangan dalam data. Proses ini secara keseluruhan mengurangi ukuran data yang dikompresi, sehingga lebih efisien dalam penggunaan bit.
2. *Move-to-Front (MTF) coding* bisa meningkatkan jumlah bit jika teks tidak banyak mengandung angka atau karakter yang sering berulang secara lokal. Hal ini disebabkan oleh ketergantungan MTF coding pada efisiensi pengkodean karakter yang sering muncul berdekatan dalam konteksnya.
3. Algoritma Sequitur menjadi lebih efektif ketika terdapat banyak pola berulang, karena algoritma ini mendekripsi pola-pola tersebut dalam data dan menggantinya dengan simbol yang lebih pendek. Hal ini menghasilkan aturan produksi yang merepresentasikan pola berulang secara lebih efisien, yang pada akhirnya mengurangi ukuran keseluruhan data yang harus disimpan atau ditransmisikan.

4. Algoritma *Move-to-Front (MTF) coding* dan algoritma Sequitur tidak akan mengubah ukuran file setelah proses dekompresi. Kedua algoritma ini dirancang untuk mengompresi data tanpa kehilangan informasi, sehingga ketika data dikembalikan ke bentuk aslinya melalui dekompresi, ukurannya akan sama seperti sebelum kompresi.
5. Semakin banyak pola kata yang berulang, kinerja algoritma Sequitur akan lebih efektif.
6. Semakin banyak angka yang muncul, kinerja algoritma MTF Coding akan lebih efektif.

5.1. Saran

Berdasarkan hasil penelitian ini, ditemukan beberapa saran yang mungkin bermanfaat dan dapat dipertimbangkan untuk penelitian lanjutan, di antaranya adalah sebagai berikut:

1. Implementasi algoritma Sequitur masih dikerjakan secara manual yang membuat kompresi akan jauh lebih sulit apabila pada file teks berisi banyak karakter. Diharapkan untuk penelitian selanjutnya implementasi dapat berupa program agar perbandingan dapat dilakukan dengan menggunakan lebih banyak perbandingan dengan teks.
2. Kompresi algoritma *Move-to-Front (MTF) coding* yang digunakan pada penelitian ini adalah algoritma *Basic Move-to-Front*. Diharapkan algoritma MTF Coding dapat lebih dikembangkan untuk pada penelitian selanjutnya.
3. Ada baiknya untuk membandingkan kompresi terhadap berbagai jenis data lainnya, seperti gambar, audio, video, dokumen, dan data komputer lainnya.

DAFTAR PUSTAKA

- Cahayati, D., Pardede, A. M. H., & Khair, H. (2022). *Implementasi Algoritma Elias Gamma Kompresi Pada File Teks.* 6341(April), 159–166.
- Darwiyanto, E. (2015). 2015 3rd International Conference on Information and Communication Technology, ICoICT 2015. In *Institute of Electrical and Electronics Engineers. Indonesia Section Institute of Electrical and Electronics Engineers.*
- Ervin. (2007). Kompresi Data Teks Menggunakan Pendekatan Grammar Compression Dengan Algoritma Sequ Itur. *Jurnal Informatika, Vol 3, No 1* (2007): *Jurnal Informatika*, 20–26. <http://ti.ukdw.ac.id/ojs/index.php/informatika/article/view/41>
- Juni Leuwarta Ompusunggu. (2010). *Penerapan kombinasi algoritma sequiter dan punctured elias code untuk kompresi file teks.* 2(2), 55–59.
- Mahesa, K. (2017). *DEKOMPRESI PADA CITRA DIGITAL.* 12(1), 948–963.
- Mansyuri, U. (2021). *KOMPRESI DATA TEKS DENGAN METODE RUN LENGTH mengurangi jumlah data dalam teks . Contoh kompresi sederhana misalnya kata.* 1(2), 102–109.
- Salomon, D. (2007). Data Compression The Complete Reference FourthEdition. In *Journal of Chemical Information and Modeling* (Vol. 53, Issue 9).
- Silaban, T. F. (2022). *Kompresi File Teks dengan Menggunakan Kombinasi Squitur dan Elias Gamma Code.* 1(3), 82–87.
- Simanjuntak, W. T. W. (2021). *Analisa Perbandingan Algortima Prediction By Partial Matching Dengan Sequitur Pada Kompresi File Teks.* 5, 221–227. <https://doi.org/10.30865/komik.v5i1.3675>
- Sunaryo, S. M., Chrisantyo, L., & Lukito, Y. (2019). *Analisis Algoritma MTF , MTF-I*

dan MTF-2 pada Burrows Wheeler Compression Algorithm. 1, 21–31.

<https://doi.org/10.21460/jutei.2018.31.148>

Wibowo, A. (2012). Kompresi Data Menggunakan Metode Huffman. *Seminar Nasional*

Teknologi Informasi & Komunikasi Terapan 2012, 2(1), 47–51.

<http://publikasi.dinus.ac.id/index.php/semantik/article/view/70>

LISTING PROGRAM

Perkenalan.cs

```

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Skripsi_Fix
{
    ///<summary>
    /// Description of MainForm.
    ///</summary>
    public partial class MainForm : Form
    {
        public MainForm()
        {
            //
            // The InitializeComponent() call is required for Windows Forms designer
            support.
            //
            InitializeComponent();

            //
            // TODO: Add constructor code after the InitializeComponent() call.
            //
        }

        void KompresibtnClick(object sender, EventArgs e)
        {
            Kompresi kompresi = new Kompresi();
            kompresi.Show();
            this.Hide();
        }
    }
}

```

```
void DekompresibtnClick(object sender, EventArgs e)
{
    Dekompresi dekompressi = new Dekompresi();
    dekompressi.Show();
    this.Hide();
}

void UtamabtnClick(object sender, EventArgs e)
{
}
}
```

Kompresi.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Diagnostics;
using System.Windows.Forms;
using System.Text;

namespace Skripsi_Fix
{
    /// <summary>
    /// Description of Kompresi.
    /// </summary>
    public partial class Kompresi : Form
    {
        public string compressedType;
        private string selectedFileC;
        private byte[] precompressedData = null;
        private byte[] compressedData = null;
        public Kompresi()
        {
            //
            // The InitializeComponent() call is required for Windows Forms designer
            support.
            //
            InitializeComponent();
            //
            // TODO: Add constructor code after the InitializeComponent() call.
            //
        }
    }
}
```

```

void UtamabtnClick(object sender, EventArgs e)
{
    MainForm mainform = new MainForm();
    mainform.Show();
    this.Hide();
}

void KompresibtnClick(object sender, EventArgs e)
{
    Kompresi kompresi = new Kompresi();
    kompresi.Show();
    this.Hide();
}

void DekompresibtnClick(object sender, EventArgs e)
{
    Dekompresi dekompressi = new Dekompresi();
    dekompressi.Show();
    this.Hide();
}

void ImportclkClick(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "Text Files | *.txt";
    openFileDialog.Title = "Select Document";
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        try
        {
            //Untuk mengambil dan menampilkan File Path
            selectedFileC = openFileDialog.FileName;
            pathC.Text = selectedFileC;

            //Untuk membaca isi dari File Path
            precompressedData =
File.ReadAllBytes(Path.Combine(Directory.GetCurrentDirectory(), selectedFileC));
        ;
    }

    //untuk menampilkan Teks dan representasi byte data dalam bentuk string,
    serta menampilkan ukuran file aslinya.
    string teksIsi = Encoding.UTF8.GetString(precompressedData);
    infofile.Text = teksIsi;
    string binaryStringC = BytesToBinaryString(precompressedData);
    infobit.Text = binaryStringC;

    //untuk menampilkan ukuran data
    infoSK.Text = DocumentSize(precompressedData);
}

```

```
        MessageBox.Show("Import berhasil!");
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error: Tidak dapat membuka dokumen. Original
error: " + ex.Message);
    }
}
```

```
void MtfbtnClick(object sender, EventArgs e)
{
    if (File.Exists(selectedFileC))
    {
        compressedType = "MTF";
        Stopwatch time = new Stopwatch();
        //Menjalankan proses Kompresi
        time.Start();
        compressedData = MTF.Compress(precompressedData);
        time.Stop();

        // menampilkan representasi byte data dalam bentuk string
        string binaryStringHC = BytesToBinaryString(compressedData);
        infoC.Text = binaryStringHC;

        //menghitung dan menampilkan ukuran dari data hasil kompresi
        infokompresi.Text = DocumentSize(compressedData);

        //perhitungan kompresi, waktu, dan tampilkan
        infopengukuran.Text
        = Measurement(precompressedData, compressedData, time);

        MessageBox.Show("kompresi selesai.");
    }
    else
    {
        MessageBox.Show("File belum di input.");
    }
}

void SavebtnClick(object sender, EventArgs e)
{
    SaveFileDialog saveFileDialog = new SaveFileDialog();
```

```

// Menentukan filter nama file berdasarkan tipe kompresi yang dipilih
if (compressedType == "MTF")
{
    saveFileDialog.Filter = "MTF File|*.mtf";
}
saveFileDialog.Title = "Save Compressed Data";

if (saveFileDialog.ShowDialog() == DialogResult.OK)
{
    string selectedFilePath = saveFileDialog.FileName;
    SaveCompressedData(compressedData, selectedFilePath);
}
}

public static string DocumentSize(byte[] data)
{
    long dataSizeInBytes = data.Length;
    double dataSizeInKB = dataSizeInBytes / 1024.0;
    string dataSize = dataSizeInKB.ToString("N2") + " KB
    (" + dataSizeInBytes.ToString("N0") + " bytes)";
    return dataSize;
}
public static string BytesToBinaryString(byte[] bytes)
{
    string binaryString = string.Join(" ", bytes.Select(b
=> Convert.ToString(b, 2).PadLeft(8, '0')));
    return binaryString;
}
public static List<byte> BinaryStringToBytes(string binaryString)
{
    List<byte> byteArray = new List<byte>();
    for (int i = 0; i < binaryString.Length; i += 8)
    {
        string byteString = binaryString.Substring(i, 8);
        byte byteValue = Convert.ToByte(byteString, 2);
        byteArray.Add(byteValue);
    }
    return byteArray;
}
public static string Measurement(byte[]{before, after}, Stopwatch time)
{
    double CR = (double)before.Length / (double)after.Length;
    double RC = (double)after.Length / (double)before.Length;
    double SS = 100 - (RC * 100);
    double RunningTime = (double)time.Elapsed.TotalMilliseconds;
    string infoText = "Ratio of Compression (RC):
" + CR.ToString("F2") + "\r\n" +
    "Compression Ratio (CR): " + RC.ToString("F2") + "\r\n" +

```

```

    "Space Savings (SS): " + SS.ToString("F2") + "%\r\n" +
    "Running Time: " + RunningTime.ToString("F2") + " milliseconds";
    return infoText;
}

public void SaveCompressedData(byte[] compressedData, string filePath)
{
    try
    {
        File.WriteAllBytes(filePath, compressedData);
        MessageBox.Show("Data berhasil disimpan ke " + filePath);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error: Gagal menyimpan data kompresi. Pesan
kesalahan: " + ex.Message);
    }
}
}

```

Dekompresi.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Diagnostics;
using System.Windows.Forms;
using System.Text;

namespace Skripsi_Fix
{
    /// <summary>
    /// Description of Dekompresi.
    /// </summary>
    public partial class Dekompresi : Form
    {
        public string compressedType;
        private string selectedFileD;
        private byte[] predecompressedData = null;
        private byte[] decompressedData = null;
        public Dekompresi()
        {
            //
        }
    }
}

```

```

// The InitializeComponent() call is required for Windows Forms designer
support.
//
InitializeComponent();

//
// TODO: Add constructor code after the InitializeComponent() call.
//
}

void UtamabtnClick(object sender, EventArgs e)
{
    MainForm mainform = new MainForm();
    mainform.Show();
    this.Hide();
}

void KompresibtnClick(object sender, EventArgs e)
{
    Kompresi kompresi = new Kompresi();
    kompresi.Show();
    this.Hide();
}

void DekompresibtnClick(object sender, EventArgs e)
{
    Dekompresi dekompressi = new Dekompresi();
    dekompressi.Show();
    this.Hide();
}

void ImportbtnClick(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "Compressed Files | *.mtf;*.sequitur;";
    openFileDialog.Title = "Select Document";
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        try
        {
            selectedFileD = openFileDialog.FileName;
            string textFromFile = File.ReadAllText(selectedFileD);
            if (selectedFileD.EndsWith(".mtf", StringComparison.Ordinal))
            {
                compressedType = "MTF";
            }
            else
            {
                MessageBox.Show("Tipe kompresi tidak valid.");
            }
        }
    }
}

```

```

        return;
    }

    // Baca file
    predecompressedData =
File.ReadAllText(Path.Combine(Directory.GetCurrentDirectory(), selectedFileD))
;

    // Mengonversi array byte ke string biner
    string binaryStringD = BytesToBinaryString(predecompressedData);

    //menampilkan path di TextBox
    pathD.Text = selectedFileD;

    // Menampilkan representasi biner dalam TextBox
    infoDF.Text = binaryStringD;

    // Menampilkan ukuran data dalam TextBox
    infobyteh.Text = DocumentSize(predecompressedData);

    MessageBox.Show("Import berhasil!");
}
catch (Exception ex)
{
    MessageBox.Show("Error: Tidak dapat membuka dokumen. Original
error: " + ex.Message);
}

void DbtnClick(object sender, EventArgs e)
{
    decompressedData = null;
    string selectedFile = pathD.Text;
    if (File.Exists(selectedFile))
    {
        Stopwatch time = new Stopwatch();

        time.Start();
        if (compressedType == "MTF")
        {
            // Jalankan proses dekompresi MTF
            decompressedData = MTF.Decompress(predecompressedData);
        }
        time.Stop();

        if (decompressedData != null)
        {

```

```

// Ubah data menjadi Binary String dan tampilkan
string binaryStringHD = BytesToBinaryString(decompressedData);
infoDbyte.Text = binaryStringHD;

// Menampilkan ukuran data
infobytea.Text = DocumentSize(decompressedData);

//Menampilkan teks
string teksIsi = Encoding.UTF8.GetString(decompressedData);
infopengukuran.Text = teksIsi;

// Menampilkan hasil pengukuran data dan waktu
infoD.Text
= Measurement(decompressedData, predecompressedData, time);

    MessageBox.Show("Dekompresi selesai.");
}
else
{
    MessageBox.Show("Tipe kompresi tidak valid.");
}
}
else
{
    MessageBox.Show("File belum di input.");
}
}

public static string DocumentSize(byte[] data)
{
    long dataSizeInBytes = data.Length;
    double dataSizeInKB = dataSizeInBytes / 1024.0;
    string dataSize = dataSizeInKB.ToString("N2") + " KB
(" + dataSizeInBytes.ToString("N0") + " bytes)";
    return dataSize;
}
public static string BytesToBinaryString(byte[] bytes)
{
    string binaryString = string.Join(" ", bytes.Select(b
=> Convert.ToString(b, 2).PadLeft(8, '0')));
    return binaryString;
}
public static List<byte> BinaryStringToBytes(string binaryString)
{
    List<byte> byteArray = new List<byte>();
    for (int i = 0; i < binaryString.Length; i += 8)
    {
        string byteString = binaryString.Substring(i, 8);
        byte byteValue = Convert.ToByte(byteString, 2);
        byteArray.Add(byteValue);
    }
    return byteArray;
}

```

```

        byteArray.Add(byteValue);
    }
    return byteArray;
}
public static string Measurement(byte[] before, byte[] after, Stopwatch time)
{
    double CR = (double)before.Length / (double)after.Length;
    double RC = (double)after.Length / (double)before.Length;
    double SS = 100 - (RC * 100);
    double RunningTime = (double)time.Elapsed.TotalMilliseconds;
    string infoText = "Ratio of Compression (RC):"
    + CR.ToString("F2") + "\r\n" +
    "Compression Ratio (CR): " + RC.ToString("F2") + "\r\n" +
    "Space Savings (SS): " + SS.ToString("F2") + "%\r\n" +
    "Running Time: " + RunningTime.ToString("F2") + " milliseconds";
    return infoText;
}

public void SaveCompressedData(byte[] compressedData, string filePath)
{
    try
    {
        File.WriteAllBytes(filePath, compressedData);
        MessageBox.Show("Data berhasil disimpan ke " + filePath);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error: Gagal menyimpan data kompresi. Pesan kesalahan: " + ex.Message);
    }
}

void SavebtndClick(object sender, EventArgs e)
{
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    saveFileDialog.Title = "Save Data";
    saveFileDialog.Filter = "Text Files|*.txt";

    if (saveFileDialog.ShowDialog() == DialogResult.OK)
    {
        string selectedFilePath = saveFileDialog.FileName;
        SaveCompressedData(decompressedData, selectedFilePath);
    }
}

}

```

MTF.cs

```

using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Linq;

namespace Skripsi_Fix
{
    ///<summary>
    /// Description of MTF.
    ///</summary>
    public class MTF
    {
        public static byte[] Compress(byte[] input)
        {
            //mengambil jumlah char unik untuk meta data untuk ukuran penyimpanan
            int uniqChar = CalculateUniqueBytesCount(input);
            int savedSize = CalculateStorageSize(uniqChar);

            //melakukan encoding dan memisahkan angka kode dan huruf kamus
            var encodedFile = Encode(input);
            byte[] encodedNumber = encodedFile.Item2;
            byte[] dictionary = encodedFile.Item1;

            //mengurangi ukuran angka kode dan menambahkan padding kemudian
            mengembalikan ke data byte
            string compressedData
            = ConvertByteArrayToBinaryString(encodedNumber,savedSize);
            byte[] compressedByte = ConvertToBytes(compressedData);

            //menyatukan seluruh data kamus, data terkompres, serta ukuran kompres
            sebagai output
            byte[] output
            = CombineByteArrays(dictionary, compressedByte, savedSize);
            return output;
        }

        public static byte[] Decompress(byte[] input)
        {
            // Ekstrak storage size dan data yang sebenarnya serta mengambil ukuran
            penyimpanan data pada metadata
            Tuple<int, byte[]> storageAndData = ExtractStorageSize(input);
            int storageSize = storageAndData.Item1;
            byte[] newData = storageAndData.Item2;
        }
    }
}

```

```

// Ekstrak dictionary dan byte terenkripsi dari data yang sebenarnya
Tuple<byte[], byte[]> dictionaryAndEncodedBytes
= ExtractDictionaryAndEncodedBytes(newData);
byte[] dictionary = dictionaryAndEncodedBytes.Item1;
byte[] compressedData = dictionaryAndEncodedBytes.Item2;

//menghapus padding serta mengembalikannya ke dalam ukuran data asli
string encodedString = RemovePadding(compressedData);
byte[] encodedBytes = ConvertToByteArray(encodedString,storageSize);

//membalikkan data untuk proses dekoding kemudian melakukan decoding
byte[] reversedByte = ReverseDictionary(encodedBytes);
byte[] output = Decode(reversedByte, dictionary);
return output;
}

static int CalculateUniqueBytesCount(byte[] bytes)
{
    HashSet<byte> uniqueSet = new HashSet<byte>(bytes);
    return uniqueSet.Count;
}
static int CalculateStorageSize(int number)
{
    return number == 0 ? 1 : (int)Math.Ceiling(Math.Log(number, 2));
}

static string ConvertByteArrayToBinaryString(byte[] byteArray, int storageSize)
{
    StringBuilder sb = new StringBuilder();

    foreach (byte b in byteArray)
    {
        int mask = (1 << storageSize) - 1; // Membuat mask dengan storageSize bit
        yang diatur menjadi 1
        int truncatedValue = b & mask; // Masking untuk memotong bit yang tidak
        diperlukan

        string binary =
Convert.ToString(truncatedValue, 2).PadLeft(storageSize, '0'); // Mengonversi ke
string biner
        sb.Append(binary); // Menambahkan ke string biner
    }

    return sb.ToString();
}
static byte[] ConvertToBytes(string binaryString)
{
    int restLength = binaryString.Length % 8;

```

```

StringBuilder stringText = new StringBuilder(binaryString);
int padding = 0;
if (restLength != 0)
{
    padding = 8 - restLength;
    for (int i = 0; i < padding; i++)
        stringText.Append("0");
}
string binaryPadding = Convert.ToString(padding, 2);
int y = 8 - binaryPadding.Length;
for (int i = 0; i < y; i++)
    stringText.Append("0");

stringText.Append(binaryPadding);

// Konversi string bit menjadi byte-byte
List<byte> bytes = new List<byte>();

for (int i = 0; i < stringText.Length; i += 8)
{
    string byteString =
stringText.ToString(i, Math.Min(8, stringText.Length - i));
    byte b = Convert.ToByte(byteString, 2);
    bytes.Add(b);
}

return bytes.ToArray();
}

static byte[] CombineByteArrays(byte[] dictionary, byte[] encodedBytes, int storageSize)
{
    List<byte> combinedBytes = new List<byte>();

    //Tambahkan storageSize sebagai metadata
    combinedBytes.Add((byte)storageSize);

    //Tambahkan dictionary
    combinedBytes.AddRange(dictionary);

    //Tambahkan pemisah
    combinedBytes.Add(0xFF);

    //Tambahkan byte yang terenkripsi
    combinedBytes.AddRange(encodedBytes);

    return combinedBytes.ToArray();
}
static Tuple<int, byte[]> ExtractStorageSize(byte[] input)

```

```

{
    int storageSize = input[0]; // Mengambil 7 bit terakhir
    byte[] newData = new byte[input.Length - 1];
    Array.Copy(input, 1, newData, 0, newData.Length);

    return Tuple.Create(storageSize, newData);
}
static Tuple<byte[], byte[]> ExtractDictionaryAndEncodedBytes(byte[] input)
{
    List<byte> dictionary = new List<byte>();
    List<byte> encodedBytes = new List<byte>();
    int currentIndex = 0;

    // Cari pemisah
    while (currentIndex < input.Length && input[currentIndex] != 0xFF)
    {
        dictionary.Add(input[currentIndex]);
        currentIndex++;
    }

    // Lewati pemisah
    currentIndex++;

    // Sisipkan sisa byte sebagai encoded bytes
    while (currentIndex < input.Length)
    {
        encodedBytes.Add(input[currentIndex]);
        currentIndex++;
    }

    return Tuple.Create(dictionary.ToArray(), encodedBytes.ToArray());
}

static string RemovePadding(byte[] input)
{
    // Konversi array byte menjadi string biner
    StringBuilder binaryString = new StringBuilder();
    foreach (byte b in input)
    {
        // Konversi byte menjadi string biner dengan panjang 8 bit dan tambahkan
        ke string
        binaryString.Append(Convert.ToString(b, 2).PadLeft(8, '0'));
    }

    // Baca panjang padding dari 8 bit terakhir
    int paddingLength =
    Convert.ToInt32(binaryString.ToString().Substring(binaryString.Length - 8, 8), 2);

    // Hapus padding dari string biner
}

```

```

    string dataWithoutPadding =
binaryString.ToString().Substring(0, binaryString.Length - 8 - paddingLength);

    return dataWithoutPadding;
}

static byte[] ConvertToByteArray(string encodedString, int storageSize)
{
    List<byte> bytes = new List<byte>();

    for (int i = 0; i < encodedString.Length; i += storageSize)
    {
        string encodedSymbol = encodedString.Substring(i, storageSize);
        byte byteValue = Convert.ToByte(encodedSymbol, 2);
        bytes.Add(byteValue);
    }

    return bytes.ToArray();
}
static byte[] ReverseDictionary(byte[] symbols)
{
    Array.Reverse(symbols);
    return symbols;
}
// Fungsi untuk dekompresi
static Tuple<byte[], byte[]> Encode(byte[] input)
{
    List<byte> encodedBytes = new List<byte>();
    List<byte> symbols = new List<byte>();

    // Inisialisasi dictionary dengan byte unik yang muncul satu kali
    HashSet<byte> uniqueBytes = new HashSet<byte>(input);
    symbols.AddRange(uniqueBytes);

    foreach (byte b in input)
    {
        int index = symbols.IndexOf(b);
        string binary = Convert.ToString(index, 2).PadLeft(8, '0'); // Menjadi
panjang 8 bit

        // Ubah string biner menjadi byte
        byte byteValue = Convert.ToByte(binary, 2);
        encodedBytes.Add(byteValue);
        // Pembaruan posisi byte dalam dictionary
        symbols.RemoveAt(index);
        symbols.Insert(0, b);
    }

    return Tuple.Create(symbols.ToArray(), encodedBytes.ToArray());
}

```

```
}

static byte[] Decode(byte[] encodedBytes, byte[] symbols)
{
    List<byte> decodedBytes = new List<byte>();
    List<byte> symbolList = new List<byte>(symbols);

    foreach (int encodedByte in encodedBytes)
    {
        byte symbol = symbolList[0];

        // Keluarkan simbol yang ditemukan dari dictionary
        symbolList.Remove(symbol);
        // Sisipkan kembali simbol ke indeks yang sesuai
        symbolList.Insert((encodedByte), symbol);

        decodedBytes.Add(symbol);
    }
    byte[] output = ReverseDictionary(decodedBytes.ToArray());
    return output;
}
```

Perhitungan Bahan Uji Algoritma Sequitur

abcabcbc.txt

String : abcabcbc

Seleksi Teks : ab,bc,ca,ab,bc,ca,ab,bc

Perulangan terjadi pada 41 maka diganti dengan $A \rightarrow ab$

Maka : AcAcAc

Seleksi : Ac,cA,Ac,cA,Ac

Perulangan terjadi pada 26 maka diganti dengan $B \rightarrow Ac$

Maka : BBB

Karena simbol A digunakan sekali dan tergantikan pada simbol $B \rightarrow Ac$, maka simbol A dihapus dan dimasukkan ke $B \rightarrow abc$.

Sehingga rule yang dihasilkan adalah BBB.

Perhitungan sebelum dikompresi

KARAKTER	FREKUENSI	DESIMAL	BINER	BIT	BIT X FREKUENSI
a	3	97	0110 0001	8	24
b	3	98	0110 0010	8	24
c	3	99	0110 0011	8	24
Jumlah	9				72

Perhitungan setelah dilakukan kompresi algoritma sequitur

KARAKTER	FREKUENSI	DESIMAL	BINER	BIT	BIT X FREKUENSI
B	3	66	0100 0010	8	24
Jumlah	3				24

$$\begin{aligned}
 RC &= 9 / 3 = 3 \\
 CR &= 3 / 9 = 0,3 \\
 SS &= 100\% - 30\% = 70\%
 \end{aligned}$$

pi.txt

String :

3.141592653589793238462643383279502884197169399375

Seleksi Teks :

3.,1,14,41,15,59,92,26,65,53,35,58,89,97,79,93,32,23,38,84,46,62,26,64,43,3

3,38,83,32,27,79,95,50,02,28,88,84,41,19,97,71,16,69,93,39,99,93,37,75

Perulangan terjadi pada 41 maka diganti dengan A → 41

Maka :

3.1A59265358979323846264338327950288A97169399375

Seleksi :

3.,1,1A,A5,59,92,26,65,53,35,58,89,97,79,93,32,23,38,84,46,62,26,64,43,33,3

8,83,32,27,79,95,50,02,28,88,8A,A9,97,71,16,69,93,39,99,93,37,75

Perulangan terjadi pada 26 maka diganti dengan B → 26

Maka :

3.1A59B5358979323846B4338327950288A97169399375

Seleksi :

3.,1,1A,A5,59,9B,B5,53,35,58,89,97,79,93,32,23,38,84,46,6B,B4,43,33,38,83,32,

27,79,95,50,02,28,88,8A,A9,97,71,16,69,93,39,99,93,37,75

Perulangan terjadi pada 97 maka di ganti dengan C → 97

Maka :

3.1A59B5358C9323846B4338327950288AC169399375

Seleksi :

3.,1,1A,A5,59,9B,B5,53,35,58,8C,C9,93,93,32,23,38,84,46,6B,B4,43,33,38,83,32

,27,79,95,50,02,28,88,8A,AC,C1,16,69,93,39,99,93,37,75

Perulangan terjadi pada 93, maka di ganti dengan D → 93

Maka : 3.1A59B5358CD23846B4338327950288AC16D9D75

Seleksi :

3.,1,1A,A5,59,9B,B5,53,35,58,8C,CD,D2,23,38,84,46,6B,B4,43,33,38,83,32,

27,79,95,50,02,28,88,8A,AC,C1,16,6D,D9,9D,D7,75

Perhitungan sebelum dikompresi

KARAKTER	FREKUENSI	DESIMAL	BINER	BIT	BIT X FREKUENSI
.	1	46	00101110	8	8
0	1	48	00110000	8	8
1	4	49	00110001	8	32
2	5	50	00110010	8	40
3	9	51	00110011	8	72
4	4	52	00110100	8	32
5	5	53	00110101	8	40
6	4	54	00110110	8	32
7	4	55	00110111	8	32
8	5	56	00111000	8	40
9	8	57	00111001	8	64
Jumlah	50				400

Perhitungan setelah dilakukan kompresi algoritma sequitur

KARAKTER	FREKUENSI	DESIMAL	BINER	BIT	BIT X FREKUENSI
.	1	46	00101110	8	8
0	1	48	00110000	8	8
1	2	49	00110001	8	16
2	3	50	00110010	8	24
3	6	51	00110011	8	48
4	2	52	00110100	8	16
5	5	53	00110101	8	40
6	2	54	00110110	8	16
7	2	55	00110111	8	16
8	5	56	00111000	8	40
9	3	57	00111001	8	24
A	2	65	01000001	8	16
B	2	66	01000010	8	16
C	2	67	01000011	8	16

D	3	68	01000100	8	24
Jumlah	41				328

$$RC = 50 / 41 = 1,22$$

$$CR = 41 / 50 = 0,82$$

$$SS = 100\% - 82\% = 18\%$$

Teks huruf acak random.txt

String : qwertajklqwerasdvxvcxvna

Seleksi Teks :

qw,we,er,rt,ta,aj,jk,kl,lq,qw,we,er,ra,as,sd,dv,vx,xc,cx,xv,vn,na

Perulangan terjadi pada qw maka diganti dengan A → qw

Maka : AertajklqAerasdvxvcxvna

Seleksi Teks :

Ae,er,rt,ta,aj,jk,kl,lq,,qA,Ae,er,ra,as,sd,dv,vx,xc,cx,xv,vn,na

Perulangan terjadi pada Ae maka diganti dengan B → Ae,

Karena simbol A digunakan sekali dan tergantikan pada simbol B → Ae, maka simbol A dihapus dan dimasukkan ke B → qwe.

Maka : BrtajklqBrasdvxvcxvna

Seleksi Teks : Br,rt,ta,aj,jk,kl,lq,qB,Br,ra,as,sd,dv,vx,xc,cx,xv,vn,na

Perulangan terjadi pada Br maka diganti dengan C → Br ,

Karena simbol B digunakan sekali dan tergantikan pada simbol C → Br, maka simbol B dihapus dan dimasukkan ke C → qwer.

Maka : CtajklqCasdvxvcxvna

Seleksi : Ct,ta,aj,jk,kl,lq,qC,Ca,as,sd,dv,vx,xc,cx,xv,vn,na

Tidak ada perulangan lagi, maka:

Perhitungan sebelum dikompresi

KARAKTER	FREKUENSI	DESIMAL	BINER	BIT	BIT X FREKUENSI
q	2	113	1110001	8	16
w	2	119	1110111	8	16
e	2	101	1100101	8	16
r	2	114	1110010	8	16
t	1	116	1110100	8	8
a	3	97	1100001	8	24
j	1	106	1101010	8	8
k	1	107	1101011	8	8
l	1	108	1101100	8	8
s	1	115	1110011	8	8
d	1	100	1100100	8	8
v	2	118	1110110	8	16
n	1	110	1101110	8	8
x	2	120	1111000	8	16
d	1	100	1100100	8	8
Jumlah	23	-	-	-	184

Perhitungan setelah dilakukan kompresi algoritma sequitur

KARAKTER	FREKUENSI	DESIMAL	BINER	BIT	BIT X FREKUENSI
C	2	67	1000011	8	16
t	1	116	1110100	8	8
a	3	97	1100001	8	24
j	1	106	1101010	8	8
k	1	107	1101011	8	8
l	1	108	1101100	8	8
q	1	113	1110001	8	8
s	1	115	1110011	8	8
d	1	100	1100100	8	8
v	2	118	1110110	8	16
x	2	120	1111000	8	16
c	1	99	1100011	8	8

n	1	110	1101110	8	8
Jumlah	18	-	-	-	144

$$RC = 23 / 18 = 1,27$$

$$CR = 18 / 23 = 0,78$$

$$SS = 100\% - 78\% = 22\%$$

CURRICULUM VITAE

DATA DIRI



Nama	: Donisius Martin Sirait
Tanggal Lahir	: 10 Maret 1999
Jenis Kelamin	: Laki-laki
Alamat	: Jl. Jahe 2 no. 2, Perumnas Simalingkar
Email	: donisiusmartin@gmail.com

RIWAYAT PENDIDIKAN

2006 – 2011	: SD Negeri 173105, Tarutung
2011 – 2014	: SMP Negeri 3, Tarutung
2014 – 2017	: SMA Negeri 1, Tarutung
2017 – 2024	: S-1 Ilmu Komputer Universitas Sumatera Utara

PENGALAMAN ORGANISASI DAN KEPANITIAAN

2019	: Anggota Game Dev ITIKOM USU
2018	: Seksi Acara Perayaan Natal S1 Ilmu Komputer USU
2019 – 2020	: Anggota Divisi PTT Kepengurusan KMKI

PENGALAMAN MAGANG

2020	: Rumah Sakit Umum Pusat Haji Adam Malik Medan
------	--

KEMAMPUAN

Pemrograman	: HTML, Pascal, C, C++, C#, Java, PHP
IDE	: NPP, CodeBlocks, SharpDevelop, Netbeans, Visual Studio, Android Studio
Basis data	: MySQL

Perangkat lunak : Ms. Office, Adobe Photoshop, Adobe Illustrator, Unity,

SEMINAR, WORKSHOP, PELATIHAN

- | | |
|------|---|
| 2018 | : Bukalapak ke Kampus |
| 2019 | : Monster Goes to Campus |
| 2020 | : Webinar International Conference on Computing and Applied Informatics (ICCAI) 2020 |
| 2021 | : Seminar Online IKA ILKOMP Sharing Discussion #1 dengan judul Kiat dan Tips Diterima di Perusahaan Multinasional |