

**PERBANDINGAN ALGORITMA *STOUT CODE* DAN ALGORITMA
FIBONACCI CODE PADA APLIKASI KOMPRESI *FILE* TEKS
BERBASIS *ANDROID***

SKRIPSI

FADHLI IBRAHIM SIREGAR

171401006



**PROGRAM STUDI S-1 ILMU KOMPUTER
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS SUMATERA UTARA
MEDAN
2023**

PERBANDINGAN ALGORITMA *STOUT CODE* DAN ALGORITMA
FIBONACCI CODE PADA APLIKASI KOMPRESI *FILE* TEKS
BERBASIS *ANDROID*

SKRIPSI

Diajukan untuk melengkapi tugas akhir dan memenuhi syarat memperoleh ijazah
Sarjana Ilmu Komputer

FADHLI IBRAHIM SIREGAR

171401006



PROGRAM STUDI S-1 ILMU KOMPUTER
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS SUMATERA UTARA
MEDAN
2023

PERSETUJUAN

Judul : PERBANDINGAN ALGORITMA *STOUT CODE* DAN
ALGORITMA *FIBONACCI CODE* PADA APLIKASI
KOMPRESI *FILE* TEKS BERBASIS *ANDROID*

Kategori : SKRIPSI

Nama : FADHLI IBRAHIM SIREGAR

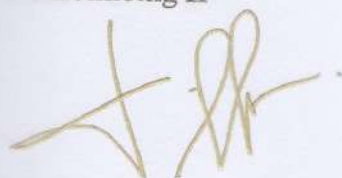
Nomor Induk Mahasiswa : 171401006

Program Studi : SARJANA (S-1) ILMU KOMPUTER

Fakultas : ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITASI SUMATERA UTARA

Komisi Pembimbing :

Pembimbing II



Dr. T. Henny Febriana Harumy, S.Kom., M.Kom.
NIP. 198802192019032016

Pembimbing I



Handrizal, S.Si., M.Comp.Sc.
NIP. 197706132017061001

Diketahui/disetujui oleh

Program Studi S-1 Ilmu Komputer



Dr. Amalia, S.T., M.T.

NIP. 197812212014042001

PERNYATAAN

**PERBANDINGAN ALGORITMA *STOUT CODE* DAN ALGORITMA *FIBONACCI CODE* PADA APLIKASI KOMPRESI *FILE* TEKS
BERBASIS *ANDROID***

SKRIPSI

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

Medan, 6 Desember 2023



Fadhli Ibrahim Siregar

171401006

PENGHARGAAN

Puji dan syukur kehadiran Allah SWT yang telah memberikan kesehatan dan kesempatan kepada penulis sehingga penulis dapat menyelesaikan skripsi yang berjudul “Perbandingan Algoritma *Stout Code* dan Algoritma *Fibonacci Code* pada Aplikasi Kompresi *File* Teks Berbasis *Android*”. Penyusunan skripsi ini merupakan salah satu syarat yang harus dipenuhi untuk memperoleh gelar Sarjana Komputer pada Program Studi S-1 Ilmu Komputer Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara.

Penyusunan skripsi ini dapat terselesaikan tidak lain karena banyaknya bantuan kepada penulis yang diberikan dari berbagai pihak. Pada kesempatan ini, penulis mengucapkan terima kasih yang sebesar-besarnya kepada semua pihak yang telah membantu penulis.

Dengan segala hormat, izinkan penulis mengucapkan terima kasih kepada:

1. Bapak Prof. Dr. Muryanto Amin, S.Sos., M.Si. selaku Rektor Universitas Sumatera Utara.
2. Ibu Dr. Maya Silvi Lydia, B.Sc., 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 Handrizal, S.Si., M.Comp.Sc. selaku Dosen Pembimbing I yang telah membimbing penulis dengan memberikan saran dan masukan beserta motivasi sehingga penulis dapat menyelesaikan skripsi ini.
5. Ibu Dr. T. Henny Febriana Harumy, S.Kom., M.Kom. selaku Dosen Pembimbing II yang telah memberikan bimbingan berupa saran dan masukan serta motivasi sehingga skripsi ini dapat terselesaikan.
6. Ibu Dian Rachmawati, S.Si., M.Kom. selaku Dosen Pembimbing Akademik.
7. Seluruh dosen dan staf pegawai di Program Studi S-1 Ilmu Komputer Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara.

8. Kedua orang tua dan kedua adik penulis yang telah mendukung secara moral dan materi serta doa untuk penulis yang selalu menyertai selama masa perkuliahan.
9. Seluruh teman mahasiswa S-1 Ilmu Komputer khususnya Kom C stambuk 2017 yang telah menemani dan membantu penulis selama masa perkuliahan.
10. Seluruh pihak yang terlibat membantu penulis baik secara langsung maupun tidak langsung yang penulis tidak dapat tuliskan satu per satu.

Medan, 6 Desember 2023

Penulis



Fadhli Ibrahim Siregar

ABSTRAK

Penelitian dalam memperkecil ukuran data terus menerus dilakukan. Kompresi data adalah proses mengubah data kedalam ukuran yang lebih kecil. Terdapat berbagai macam algoritma kompresi data dan pada penelitian ini akan digunakan algoritma *Stout Code* dan *Fibonacci Code*. Penulis melakukan pengujian kompresi *file* teks menggunakan kedua algoritma ini untuk membandingkan kinerja kedua algoritma tersebut. Parameter perbandingan yang akan digunakan adalah rasio kompresi, waktu kompresi, dan waktu dekompresi. Alat yang digunakan untuk membandingkan adalah Aplikasi berbasis *Android*. Hasil pengujian menunjukkan bahwa berdasarkan rasio kompresi, algoritma *Stout Code* didapatkan lebih baik dengan rata-rata 1,99 untuk *string* homogen dan 1,2 untuk *string* heterogen sementara algoritma *Fibonacci Code* memiliki rata-rata 1,99 untuk *string* homogen dan 1,1 untuk *string* heterogen. Berdasarkan waktu kompresi, algoritma *Fibonacci Code* didapatkan lebih baik dengan rata-rata 13 ms untuk *string* homogen dan 24,14 ms untuk *string* heterogen sementara algoritma *Stout Code* memiliki rata-rata 14,71 ms untuk *string* homogen dan 25 ms untuk *string* heterogen. Berdasarkan waktu dekompresi, algoritma *Fibonacci Code* juga didapatkan lebih baik dengan rata-rata 40,14 ms untuk *string* homogen dan 100,71 ms untuk *string* heterogen sementara algoritma *Stout Code* memiliki rata-rata 320,71 ms untuk *string* homogen dan 517,43 ms untuk *string* heterogen. Secara umum, pada pengujian kompresi *file* teks homogen dan heterogen didapatkan bahwa algoritma *Stout Code* lebih unggul dari segi pengecilan ukuran sementara algoritma *Fibonacci Code* lebih unggul dari segi kecepatan.

Kata kunci: *Stout Code*, *Fibonacci Code*, Kompresi, Dekompresi, *File* Teks, *Android*

COMPARISON OF STOUT CODE ALGORITHM AND FIBONACCI CODE ALGORITHM IN ANDROID-BASED TEXT FILE COMPRESSION APPLICATION

ABSTRACT

Research in reducing data size is continuously being carried out. Data compression is the process of converting data into smaller size. There are various data compression algorithms. In this study, the Stout Code and Fibonacci Code algorithms will be used. The author conducted a text file compression test using these two algorithms to compare the performance of the two algorithms. The comparison parameters that will be used are compression ratio, compression time, and decompression time. The tool used to compare is an Android-based application. The test results show that based on the compression ratio, the Stout Code algorithm is better with an average of 1,99 for homogeneous strings and 1,2 for heterogeneous strings while the Fibonacci Code algorithm has an average of 1,99 for homogeneous strings and 1,1 for heterogeneous strings. Based on the compression time, the Fibonacci Code algorithm is better with an average of 13 ms for homogeneous strings and 24,14 ms for heterogeneous strings while the Stout Code algorithm has an average of 14,71 ms for homogeneous strings and 25 ms for heterogeneous strings. Based on the decompression time, the Fibonacci Code algorithm is also better with an average of 40,14 ms for homogeneous strings and 100,71 ms for heterogeneous strings while the Stout Code algorithm has an average of 320,71 ms for homogeneous strings and 517,43 ms for heterogeneous strings. In general, in the compression test of homogeneous and heterogeneous text files, it is found that the Stout Code algorithm is superior in terms of size reduction, while the Fibonacci Code algorithm is superior in terms of speed.

Keywords: Stout Code, Fibonacci Code, Compression, Decompression, Text file, Android

DAFTAR ISI

PERSETUJUAN	ii
PERNYATAAN	iii
PENGHARGAAN	iv
ABSTRAK.....	vi
ABSTRACT.....	vii
DAFTAR ISI.....	viii
DAFTAR TABEL.....	xi
DAFTAR GAMBAR	xii
BAB 1 PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Rumusan Masalah	2
1.3. Batasan Masalah.....	2
1.4. Tujuan Penelitian.....	2
1.5. Manfaat Penelitian.....	3
1.6. Metodologi Penelitian	3
1.7. Sistematika Penulisan.....	4
BAB 2 LANDASAN TEORI.....	5
2.1. Kompresi Data.....	5
2.1.1. Definisi Kompresi Data	5
2.1.2. Teknik Kompresi Data	5
2.2. <i>File</i> Teks.....	6
2.3. Algoritma <i>Stout Code</i>	6
2.4. Algoritma <i>Fibonacci Code</i>	11
2.5. Parameter Analisis Kinerja Kompresi	14

2.6. <i>Android</i>	14
2.6.1. Aplikasi Berbasis Android	15
2.7. Penelitian yang Relevan	15
BAB 3 ANALISIS DAN PERANCANGAN	16
3.1. Analisis Sistem.....	16
3.1.1 Analisis Masalah	16
3.1.2 Analisis Kebutuhan	17
3.2. Pemodelan Sistem	18
3.2.1 Arsitektur Umum	18
3.2.2 Use case Diagram	19
3.2.3 Activity Diagram	21
3.2.4 Sequence Diagram	23
3.3. <i>Flowchart</i>	25
3.3.1 Flowchart Sistem	25
3.3.2 Flowchart Stout Code.....	28
3.3.3 Flowchart Fibonacci Code	29
3.4 Perancangan Antarmuka.....	30
BAB 4 IMPLEMENTASI DAN PENGUJIAN SISTEM	34
4.1 Implementasi Sistem	34
4.1.1 Fragment Kompresi	34
4.1.2 Fragment Dekompresi.....	35
4.2 Pengujian Sistem	35
4.2.1 Pengujian Proses Kompresi	35
4.2.2 Pengujian Proses Dekompresi.....	37
4.2.3 Hasil Pengujian	39
4.2.4 Pengujian String Homogen	41

4.2.5 Pengujian String Heterogen	45
BAB 5 KESIMPULAN DAN SARAN	49
5.1 Kesimpulan.....	49
5.2 Saran.....	50
DAFTAR PUSTAKA	51
LAMPIRAN.....	53

DAFTAR TABEL

Tabel 2.1 Himpunan Karakter (<i>Charset</i>)	7
Tabel 2.2 Pengkodean dengan <i>Stout Code</i>	10
Tabel 2.3 Konversi Karakter Menjadi <i>Stout Code</i>	10
Tabel 2.4 <i>Fibonacci Code</i>	12
Tabel 2.5 Pengkodean dengan <i>Fibonacci Code</i>	13
Tabel 2.6 Konversi Karakter Menjadi <i>Fibonacci Code</i>	13
Tabel 3.1. Keterangan Sketsa Halaman Kompresi	31
Tabel 3.2. Keterangan Sketsa Halaman Dekompresi	33
Tabel 4.1. Sampel Pengujian <i>String</i> Homogen	40
Tabel 4.2. Sampel Pengujian <i>String</i> Heterogen	40
Tabel 4.3. Hasil Pengujian <i>String</i> Homogen dengan <i>Stout Code</i>	41
Tabel 4.4. Hasil Pengujian <i>String</i> Homogen dengan <i>Fibonacci Code</i>	41
Tabel 4.5. Hasil Pengujian <i>String</i> Heterogen dengan <i>Stout Code</i>	45
Tabel 4.6. Hasil Pengujian <i>String</i> Heterogen dengan <i>Fibonacci Code</i>	45

DAFTAR GAMBAR

Gambar 3.1. Diagram <i>Ishikawa</i>	16
Gambar 3.2. Arsitektur Umum	19
Gambar 3.3. <i>Use case</i> Diagram	20
Gambar 3.4. <i>Activity</i> Diagram Proses Kompresi	21
Gambar 3.5. <i>Activity</i> Diagram Proses Dekompresi.....	22
Gambar 3.6. <i>Sequence</i> Diagram Proses Kompresi	23
Gambar 3.7. <i>Sequence</i> Diagram Proses Dekompresi.....	24
Gambar 3.8. <i>Flowchart</i> Proses Kompresi	26
Gambar 3.9. <i>Flowchart</i> Proses Dekompresi	27
Gambar 3.10. <i>Flowchart</i> Pembangkitan <i>Stout Code</i>	28
Gambar 3.11. <i>Flowchart</i> Pembangkitan <i>Fibonacci Code</i>	29
Gambar 3.12. Sketsa Halaman Kompresi	30
Gambar 3.13. Sketsa Halaman Dekompresi	32
Gambar 4.1. <i>Fragment</i> Kompresi	34
Gambar 4.2. <i>Fragment</i> Dekompresi	35
Gambar 4.3. Pengujian Kompresi	36
Gambar 4.4. Kompresi Berhasil.....	37
Gambar 4.5. Pengujian Dekompresi	38
Gambar 4.6. Dekompresi Berhasil.....	39
Gambar 4.7. Diagram Perbandingan Rasio Kompresi pada <i>String</i> Homogen.....	42
Gambar 4.8. Diagram Perbandingan Waktu Kompresi pada <i>String</i> Homogen	43
Gambar 4.9. Diagram Perbandingan Waktu Dekompresi pada <i>String</i> Homogen	44
Gambar 4.10. Diagram Perbandingan Rasio Kompresi pada <i>String</i> Heterogen.....	46
Gambar 4.11. Diagram Perbandingan Waktu Kompresi pada <i>String</i> Heterogen	47
Gambar 4.12. Diagram Perbandingan Waktu Dekompresi pada <i>String</i> Heterogen.....	48

BAB 1

PENDAHULUAN

1.1. Latar Belakang

Aktivitas manusia di zaman ini pada umumnya memiliki hubungan dengan penggunaan komputer dan internet. Ketika menggunakan komputer dalam melakukan suatu aktivitas, maka dibutuhkan penyimpanan data. Baik berupa penyimpanan *cloud*, maupun berupa penyimpanan fisik, permasalahan ukuran data menjadi hal penting terhadap efisiensi sumber daya dalam melakukan aktivitas. Oleh karena itu, penelitian dalam mencoba memperkecil ukuran data terus menerus dilakukan.

Kompresi data diartikan sebagai proses mengubah data masukan menjadi data keluaran dengan ukuran yang mengecil (Sitio, 2018). Teknik dalam melakukan kompresi data berperan penting dalam menentukan seberapa besar ukuran yang berkurang (Jayasankar et al., 2018). Dengan teknik kompresi yang efektif, akan didapatkan ukuran data yang lebih kecil dari sebelumnya. Terdapat berbagai macam algoritma yang telah dikembangkan dalam permasalahan ini. Dua diantaranya menggunakan *Stout Code* dan *Fibonacci Code*.

Stout Code merupakan algoritma yang bersifat rekursif yang ditemukan oleh Quentin Stout pada tahun 1980. Algoritma ini memiliki dua keluarga. Keluarga pertama disebut dengan R_ℓ dan yang kedua disebut dengan S_ℓ . Parameter penentu dari algoritma ini adalah satu bilangan integer yang bernilai lebih besar atau sama dengan dua. Parameter ini disebut dengan ℓ . Keluarga S_ℓ memiliki beberapa kelebihan dari keluarga R_ℓ untuk nilai ℓ yang kecil (Nasution, 2019). Pada penelitian (Nasution, 2019) didapatkan bahwa algoritma ini sesuai untuk melakukan kompresi pada teks.

Algoritma *Fibonacci Code* merupakan algoritma yang metode kompresinya menggunakan barisan Fibonacci yang diubah ke dalam biner untuk membentuk kodenya (Bhattacharyya, 2017). Pada penelitian (Rachmawati, et al., 2018) didapatkan bahwa algoritma *Fibonacci Code* lebih baik daripada *Even-Rodeh Code* dalam mengkompresi teks baik pada karakter homogen maupun heterogen.

Berdasarkan penelitian-penelitian sebelumnya, telah ditemukan bahwa algoritma *Stout Code* dan algoritma *Fibonacci Code* efektif untuk melakukan kompresi *file* teks. Oleh karena itu dalam penelitian ini, penulis ingin mengkaji perbandingan kedua algoritma menggunakan aplikasi berbasis *Android* sehingga dapat diketahui efisiensi kinerja kedua algoritma tersebut. Parameter perbandingan yang digunakan adalah waktu kompresi, waktu dekompresi dan rasio kompresi (*Compression ratio*).

1.2. Rumusan Masalah

Tuntutan akan efisiensi penyimpanan dan percepatan transfer data menyebabkan perlunya penelitian dalam mengkaji perbandingan algoritma kompresi data, sehingga diperlukan perbandingan algoritma *Stout Code* dan *Fibonacci Code* dalam melakukan kompresi *file* teks untuk mengetahui keefisienan kedua algoritma dengan bantuan aplikasi berbasis *Android*.

1.3. Batasan Masalah

Peneliti telah mempersempit cakupan masalah yang akan diteliti dalam penelitian ini. Batasan tersebut adalah:

1. Data yang akan dikecilkan ukurannya melalui proses kompresi adalah karakter ASCII dalam *file* teks yang berekstensi *.txt.
2. Ukuran data yang akan dikecilkan berada pada interval 200 bytes sampai dengan 2 MB.
3. Parameter perbandingan yang digunakan adalah waktu kompresi (dalam satuan *millisecond* (ms)) dan rasio kompresi (C_R).
4. *Kotlin* dipilih sebagai bahasa pemrograman yang digunakan dalam pembuatan aplikasi.
5. Sistem operasi yang digunakan untuk menjalankan aplikasi adalah *Android* versi 5.0 sampai dengan terbaru.

1.4. Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk membuat aplikasi berbasis *Android* yang dapat melakukan kompresi dan dekompresi *file* .txt dengan menggunakan algoritma *Stout Code* dan *Fibonacci Code* serta menunjukkan hasil perbandingan kinerja kedua algoritma berdasarkan rasio kompresi, waktu kompresi dan waktu dekompresi.

1.5. Manfaat Penelitian

Penelitian ini dapat memberikan manfaat berupa:

1. Mengetahui hasil perbandingan kinerja algoritma *Stout Code* dan *Fibonacci Code* pada aplikasi berbasis *Android*.
2. Menghasilkan aplikasi yang mampu melakukan proses kompresi dan dekompresi pada *file* teks dengan menggunakan algoritma *Stout Code* dan *Fibonacci Code*.
3. Penelitian diharapkan bermanfaat dalam upaya mengoptimalkan penyimpanan dan kecepatan transfer data ataupun dijadikan sebagai referensi pada topik tersebut.

1.6. Metodologi Penelitian

Berikut adalah urutan tahapan yang dilakukan dalam penelitian ini.

1. Studi Literatur

Pada bagian studi literatur, dilakukan proses mencari dan mengumpulkan informasi dan referensi yang diperlukan. Referensi yang digunakan dalam penelitian ini dapat berupa sumber dari berbagai media, seperti buku, jurnal, artikel di media cetak, dan juga situs internet yang relevan.

2. Analisis dan Perancangan

Analisis dilakukan dari hasil studi literatur dan kemudian dirancang aplikasi yang akan dibuat. Hasil rancangan aplikasi berupa *flowchart* (diagram alir), *Unified Modelling Language*, dan *user interface*.

3. Implementasi

Implementasi berupa aplikasi dibuat berdasarkan hasil rancangan. Aplikasi tersebut dibuat berbasis *Android* dengan *Kotlin* dipilih sebagai bahasa pemrogramannya.

4. Pengujian

Tahap ini bertujuan untuk menguji kecocokan aplikasi yang sudah dibuat.

5. Dokumentasi

Tahap ini melibatkan pembuatan skripsi sebagai dokumentasi hasil dari penelitian yang telah dilakukan.

1.7. Sistematika Penulisan

Proposal ini terdiri dari beberapa bab yang mengikuti topik yang dibahas, dan disusun dengan sistematika sebagai berikut:

BAB 1: PENDAHULUAN

Isi dari bab ini mencakup uraian mengenai latar belakang pemilihan judul "Perbandingan Algoritma *Stout Code* dan Algoritma *Fibonacci Code* pada Aplikasi Kompresi *File Teks Berbasis Android*", rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metode penelitian, dan juga sistematika penulisan.

BAB 2: LANDASAN TEORI

Isi dari bab ini mencakup uraian teori yang menjelaskan algoritma *Stout Code* dan algoritma *Fibonacci Code* pada kompresi data.

BAB 3: ANALISIS DAN PERANCANGAN

Isi dari bab ini mencakup arsitektur umum dalam perancangan aplikasi *Android* yang mengimplementasikan algoritma *Stout Code* dan *Fibonacci Code*.

BAB 4: IMPLEMENTASI DAN PENGUJIAN SISTEM

Bab ini berisi hasil penelitian berupa penjelasan mengenai aplikasi dan pengujian terhadap algoritma *Stout Code* dan *Fibonacci Code* pada aplikasi tersebut.

BAB 5: KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan yang didapatkan dari hasil penelitian dan saran berupa kelemahan dari penelitian sebagai rekomendasi untuk penelitian selanjutnya.

BAB 2

LANDASAN TEORI

2.1. Kompresi Data

2.1.1. Definisi Kompresi Data

Kompresi data adalah ilmu atau seni dalam menyajikan informasi menjadi format yang ringkas. Kompresi data dilakukan dengan cara mengidentifikasi dan menggunakan struktur yang ada dalam data. Kompresi data dibutuhkan karena semakin banyaknya informasi yang manusia hasilkan maupun simpan dalam bentuk digital (Sayoud, 2018).

2.1.2. Teknik Kompresi Data

Secara umum, teknik kompresi data dapat dibagi menjadi dua jenis, yaitu kompresi data yang kehilangan informasi (*lossy*) dan kompresi data tanpa kehilangan informasi (*lossless*).

Teknik kompresi *lossy* merupakan teknik kompresi yang melibatkan kehilangan sebagian informasi. Kompresi jenis ini tidak dapat memulihkan atau merekonstruksi data secara tepat. Akan tetapi data yang dikompresi menggunakan teknik ini umumnya dapat memperoleh rasio kompresi yang jauh lebih tinggi daripada menggunakan teknik kompresi *lossless* (Sayoud, 2018).

Kompresi multimedia merupakan penerapan paling umum pada teknik kompresi *lossy*. File suara, gambar, video, dan model tiga dimensi berkualitas tinggi biasanya berukuran sangat besar pada versi orginalnya. Di sisi lain, kehilangan beberapa *pixel* dari versi aslinya tidak membuat data menjadi tidak terbaca. Oleh karena itu, dibutuhkan teknik kompresi *lossy*.

Teknik kompresi *lossless* seperti namanya, tidak melibatkan kehilangan informasi. Jika suatu data telah dikompresi menggunakan teknik kompresi ini, maka data aslinya dapat dipulihkan kembali seperti semula tanpa kehilangan data sedikitpun. Umumnya, kompresi *lossless* dilakukan terhadap kasus - kasus dimana tidak dapat diterima adanya perbedaan antara data asli dan data yang sudah dikompresi ulang (Sayoud, 2018).

Kompresi teks adalah contoh area penting untuk kompresi *lossless*. Dalam kompresi teks, sangat penting bahwa proses rekonstruksi dapat menghasilkan teks yang identik dengan teks asli. Hal ini dikarenakan perbedaan yang sangat kecil pada suatu teks dapat menghasilkan pernyataan dengan makna yang sangat berbeda (Sayoud, 2018).

2.2. File Teks

File teks merupakan *file* yang berisi kumpulan karakter atau *string* yang bersatu menjadi sebuah unit (Tanjung & Nasution, 2020). Jenis dari *file* ini adalah *file* digital yang *non-executable* (tidak dapat dijalankan) yang isinya bisa berupa huruf, angka, simbol dan/atau kombinasi. *File* ini memungkinkan pembuatan dan penyimpanan teks tanpa pemformatan khusus apa pun (Rajasekaran, et. al., 2022).

File teks disimpan dalam bentuk yang dapat dibaca manusia, biasanya format numerik alfabet seperti *ASCII* (Venkata, et. al., 2020). Salah satu format *file* teks yang paling mendasar dan kompatibel secara luas adalah *.txt*. Karena sebagian besar berisi teks yang belum diformat, *file .txt* dapat dibuka di hampir semua jenis sistem operasi di berbagai perangkat keras.

2.3. Algoritma Stout Code

Pada tahun 1980, Quentin Stout menemukan algoritma yang disebut *Stout Code*. *Codeword* yang dihasilkan oleh algoritma *Stout Code* tergantung kepada sebuah parameter ℓ yang dipilih dengan syarat lebih besar dari atau sama dengan dua (Nasution, 2019).

Pada keluarga R_ℓ , *prefix* didefinisikan sebagai:

$$R_\ell(n) = B(n, \ell), \text{ untuk } 0 \leq n \leq 2^\ell - 1$$

$$R_\ell(n) = R_\ell(L)B(n, \ell), \text{ untuk } n \geq 2^\ell$$

$B(n, \ell)$ merupakan nilai biner n yang diambil sebanyak ℓ bit. Misalnya $B(1, 3)$ akan menghasilkan nilai biner 1 yang diambil sebanyak 3 bit yaitu 001.

L merupakan banyaknya digit pada nilai biner n . Misalnya n adalah 4, maka nilai biner n adalah 100. Terdapat tiga digit pada nilai biner n , maka nilai L adalah 3.

Codeword untuk keluarga S_ℓ dibentuk dengan metode yang sama dengan menggunakan *prefix* yang berbeda. *Prefix* ini dinotasikan dengan $S_\ell(n)$. Keluarga S_ℓ memiliki keunggulan dari keluarga R_ℓ untuk nilai ℓ yang kecil (Nasution, 2019).

Pada keluarga S_ℓ , *prefix* didefinisikan sebagai:

$$S_\ell(\mathbf{n}) = \mathbf{B}(\mathbf{n}, \ell), \text{ untuk } 0 \leq n \leq 2^\ell - 1$$

$$S_\ell(\mathbf{n}) = \mathbf{R}_\ell(\mathbf{L} - \mathbf{1} - \ell)\mathbf{B}(\mathbf{n}, \ell), \text{ untuk } n \geq 2^\ell$$

Pada rumus tersebut, dapat dilihat bahwa fase rekursif dari $S_\ell(n)$ memanggil keluarga R_ℓ yaitu pada $\mathbf{R}_\ell(\mathbf{L} - \mathbf{1} - \ell)$. Untuk lebih mudah dalam memahaminya, akan dijelaskan dengan contoh berikut ini.

Contoh:

Kompres *string* berikut ini dengan *Stout Code* keluarga S_ℓ : “FADHLI SIREGAR”

Penyelesaian:

Tentukan nilai ℓ sesuai keinginan, misal $\ell = 2$

Buat tabel *charset* dan hitung jumlah bit *string* sebelum dikompresi.

$$|\text{char}| = 12$$

$$\Sigma = \{F, A, D, H, L, I, \text{space}, S, R, E, G\}$$

Tabel 2.1 Himpunan Karakter (*Charset*)

n	Karakter	Frekuensi	ASCII Binary	Bit	Frekuensi x Bit
1	F	1	01000110	8	8
2	A	2	01000001	8	16
3	D	1	01000100	8	8
4	H	1	01001000	8	8
5	L	1	01001100	8	8
6	I	2	01001001	8	16
7	space	1	00100000	8	8
8	S	1	01010011	8	8
9	R	2	01010010	8	16
10	E	1	01000101	8	8
11	G	1	01000111	8	8
Total Bit					112

Tabel 2.1 di atas merupakan tabel himpunan karakter dan kode biner ASCII untuk *string* “FADHLI SIREGAR”.

Kemudian bangkitkan kode sesuai dengan rumus.

a. Untuk $0 \leq n \leq 2^\ell - 1$

$$2^\ell - 1 = 2^2 - 1 = 3$$

- $n = 1$

$$S_\ell(n) = B(n, \ell)$$

$$S_\ell(1) = B(1, 2) = 01$$

- $n = 2$

$$S_\ell(n) = B(n, \ell)$$

$$S_\ell(2) = B(2, 2) = 10$$

- $n = 3$

$$S_\ell(n) = B(n, \ell)$$

$$S_\ell(3) = B(3, 2) = 11$$

b. Untuk $n \geq 2^\ell$

- $n = 4$

$$S_\ell(n) = R_\ell(L - 1 - \ell)B(n, \ell)$$

$$\blacksquare R_\ell(L - 1 - \ell) = R_\ell(3 - 1 - 2) = R_\ell(0) = 00$$

$$\blacksquare B(n, \ell) = B(4, 2) = 100$$

$$S_\ell(4) = R_\ell(0)B(4, 2) = 00100$$

- $n = 5$

$$S_\ell(n) = R_\ell(L - 1 - \ell)B(n, \ell)$$

$$\blacksquare R_\ell(L - 1 - \ell) = R_\ell(3 - 1 - 2) = R_\ell(0) = 00$$

$$\blacksquare B(n, \ell) = B(5, 2) = 101$$

$$S_\ell(5) = R_\ell(0)B(5, 2) = 00101$$

- $n = 6$

$$S_\ell(n) = R_\ell(L - 1 - \ell)B(n, \ell)$$

$$\blacksquare R_\ell(L - 1 - \ell) = R_\ell(3 - 1 - 2) = R_\ell(0) = 00$$

$$\blacksquare B(n, \ell) = B(6, 2) = 110$$

$$S_\ell(6) = R_\ell(0)B(6, 2) = 00110$$

- $n = 7$

$$S_\ell(n) = R_\ell(L - 1 - \ell)B(n, \ell)$$

$$\blacksquare R_\ell(L - 1 - \ell) = R_\ell(3 - 1 - 2) = R_\ell(0) = 00$$

$$\blacksquare B(n, \ell) = B(7, 2) = 111$$

$$S_\ell(7) = R_\ell(0)B(7, 2) = 00111$$

- $n = 8$

$$S_\ell(n) = R_\ell(L - 1 - \ell)B(n, \ell)$$

$$\blacksquare R_\ell(L - 1 - \ell) = R_\ell(4 - 1 - 2) = R_\ell(1) = 01$$

$$\blacksquare B(n, \ell) = B(8, 2) = 1000$$

$$S_\ell(8) = R_\ell(0)B(8, 2) = 011000$$

- $n = 9$

$$S_\ell(n) = R_\ell(L - 1 - \ell)B(n, \ell)$$

$$\blacksquare R_\ell(L - 1 - \ell) = R_\ell(4 - 1 - 2) = R_\ell(1) = 01$$

$$\blacksquare B(n, \ell) = B(9, 2) = 1001$$

$$S_\ell(9) = R_\ell(0)B(9, 2) = 011001$$

- $n = 10$

$$S_\ell(n) = R_\ell(L - 1 - \ell)B(n, \ell)$$

$$\blacksquare R_\ell(L - 1 - \ell) = R_\ell(4 - 1 - 2) = R_\ell(1) = 01$$

$$\blacksquare B(n, \ell) = B(10, 2) = 1010$$

$$S_\ell(10) = R_\ell(0)B(10, 2) = 011010$$

- $n = 11$

$$S_\ell(n) = R_\ell(L - 1 - \ell)B(n, \ell)$$

$$\blacksquare R_\ell(L - 1 - \ell) = R_\ell(4 - 1 - 2) = R_\ell(1) = 01$$

$$\blacksquare B(n, \ell) = B(11, 2) = 1011$$

$$S_\ell(11) = R_\ell(0)B(11, 2) = 011011$$

Urutkan karakter berdasarkan frekuensi mulai dari yang terbesar menuju yang terkecil. Kemudian ganti kode biner ASCII karakter – karakter tersebut dengan *Stout Code* yang telah dibangkitkan secara berurut.

Tabel 2.2 Pengkodean dengan *Stout Code*

n	Karakter	Frekuensi	<i>Stout Code</i>	Bit	Frekuensi x Bit
1	A	2	01	2	4
2	I	2	10	2	4
3	R	2	11	2	4
4	F	1	00100	5	5
5	D	1	00101	5	5
6	H	1	00110	5	5
7	L	1	00111	5	5
8	space	1	011000	6	6
9	S	1	011001	6	6
10	E	1	011010	6	6
11	G	1	011011	6	6
Total Bit					56

Tabel 2.2 di atas menunjukkan representasi *Stout Code* untuk setiap karakter yang ada pada *string*.

Selanjutnya tukar kode biner ASCII setiap karakter pada *string* “FADHLI SIREGAR” sesuai dengan *Stout Code* yang telah ditentukan pada tabel.

Tabel 2.3 Konversi Karakter Menjadi *Stout Code*

F	A	D	H	L	I	space
00100	01	00101	00110	00111	10	011000
S	I	R	E	G	A	R
011001	10	11	011010	011011	01	11

Tabel 2.3 di atas menunjukkan penukaran setiap karakter pada *string* “FADHLI SIREGAR” dengan *Stout Code* yang bersangkutan.

Maka didapat *string bit*:

00100010010100110001111001100001100110110110100110110111

Setelah itu, tambahkan *padding bit* yang diikuti dengan *flag bit*.

Dalam kasus ini, bit pada *string bit* berjumlah 56. Karena jumlah bit pada *string bit* habis dibagi 8, maka jumlah bit pada *padding bit* adalah 0 atau tidak perlu *padding bit* sama sekali.

Flag bit berupa 8 bit nilai biner daripada jumlah bit pada *padding bit* yaitu 00000000.

Maka hasil setelah dikompresi adalah:

0010001001010011000111100110000110011011011010011011011100000000

2.4. Algoritma *Fibonacci Code*

Bilangan Fibonacci merupakan bilangan yang namanya berasal dari seorang ilmuwan Italia yaitu Leonardo Fibonacci. Bilangan – bilangan ini apabila disusun secara berurutan membentuk barisan Fibonacci. Barisan Fibonacci dapat didefinisikan sebagai barisan bilangan yang suku-sukunya adalah hasil penjumlahan dari dua suku sebelumnya. Contoh barisan Fibonacci:

1, 1, 2, 3, 5, 8, 13, ...

Algoritma *Fibonacci Code* merupakan algoritma yang metode kompresinya menggunakan barisan Fibonacci yang diubah ke dalam biner untuk membentuk kodenya. Pembentukan kode Fibonacci didasarkan pada fakta bahwa bilangan bulat positif n dapat dinyatakan secara unik sebagai jumlah dari bilangan Fibonacci yang berbeda (Bhattacharyya, 2017). Barisan Fibonacci yang digunakan pada algoritma tidak diawali dengan dua buah 1, sehingga menjadi:

1, 2, 3, 5, 8, 13, 21, ...

Misalnya kode Fibonacci untuk 4 adalah 1011. Pada kode tersebut dapat dilihat bahwa angka 1 di bagian paling kiri kode merepresentasikan bilangan 1 di barisan Fibonacci. Kemudian angka 0 merepresentasikan bilangan 2 dan angka 1 berikutnya merepresentasikan bilangan 3. Apabila bilangan yang mendapat representasi angka 1 dijumlahkan, maka hasilnya adalah 1 ditambah 3 yaitu 4. Angka 1 tambahan di akhir kode dibutuhkan dalam proses *decoding* menjadi penanda ujung akhir kode Fibonacci dari suatu karakter.

Tabel 2.4 *Fibonacci Code*

n	<i>Fibonacci Code</i>	n	<i>Fibonacci Code</i>
1	11	7	01011
2	011	8	000011
3	0011	9	100011
4	1011	10	010011
5	00011	11	001011
6	10011	12	101011

Tabel 2.4 di atas menunjukkan kode pada *Fibonacci Code* dengan n mulai dari 1 sampai 12.

Contoh:

Kompres *string* berikut ini dengan menggunakan *Fibonacci Code*:

“FADHLI SIREGAR”

Penyelesaian:

Buat tabel *charset* dan hitung jumlah bit *string* sebelum dikompresi.

$|\text{char}| = 12$

$\Sigma = \{F, A, D, H, L, I, \text{space}, S, R, E, G\}$

Tabel himpunan karakter (*charset*) dapat dilihat pada tabel 2.1 bagian contoh kompresi algoritma *Stout Code*.

Urutkan karakter berdasarkan frekuensi mulai dari yang terbesar menuju yang terkecil.

Kemudian ganti kode biner ASCII karakter – karakter tersebut dengan *Fibonacci Code*.

Tabel 2.5 Pengkodean dengan *Fibonacci Code*

n	Karakter	Frekuensi	<i>Fibonacci Code</i>	Bit	Frekuensi x Bit
1	A	2	11	2	4
2	I	2	011	3	6
3	R	2	0011	4	8
4	F	1	1011	4	4
5	D	1	00011	5	5
6	H	1	10011	5	5
7	L	1	01011	5	5
8	space	1	000011	6	6
9	S	1	100011	6	6
10	E	1	010011	6	6
11	G	1	001011	6	6
Total Bit					61

Tabel 2.5 di atas menunjukkan representasi *Fibonacci Code* untuk setiap karakter yang ada pada *string* “FADHLI SIREGAR”.

Selanjutnya tukar kode biner ASCII setiap karakter pada *string* “FADHLI SIREGAR” sesuai dengan *Fibonacci Code* yang telah ditentukan pada tabel.

Tabel 2.6 Konversi Karakter Menjadi *Fibonacci Code*

F	A	D	H	L	I	space
1011	11	00011	10011	01011	011	000011
S	I	R	E	G	A	R
100011	011	0011	010011	001011	11	0011

Tabel 2.6 di atas menunjukkan penukaran setiap karakter pada *string* “FADHLI SIREGAR” dengan *Fibonacci Code* yang bersangkutan.

Maka didapat *string bit*:

1011110001110011010110110000111000110110011010011001011110011

Setelah itu, tambahkan *padding bit* yang diikuti dengan *flag bit*.

Dalam kasus ini, bit pada *string bit* berjumlah 61. Karena jumlah bit pada *string bit* ketika dibagi 8 menyisakan 5, maka dibutuhkan 3bit tambahan. Sehingga *padding bit* adalah 000.

Flag bit berupa 8bit nilai biner dari jumlah bit pada *padding bit* yaitu 00000011.

Maka hasil setelah dikompresi adalah:

101111000111001101011011000011100011011001101001100101111001100000000
011

2.5. Parameter Analisis Kinerja Kompresi

Dalam proses kompresi data, ada parameter - parameter yang dapat berguna dalam menilai kinerja metode kompresi, yaitu:

1. Rasio Kompresi (*Compression ratio*)

Perbandingan seluruh bit yang digunakan untuk membentuk data pada saat sebelum dikompresi terhadap data sesudah dikompresi disebut rasio kompresi (Sayoud, 2017). Berikut adalah rumus rasio kompresi.

$$\text{Rasio Kompresi} = \frac{\text{Ukuran data sebelum dikompresi}}{\text{Ukuran data setelah dikompresi}}$$

Rasio kompresi juga dapat direpresentasikan dengan menunjukkan pengurangan terhadap ukuran data yang dibutuhkan sebagai persentase dari ukuran data yang asli (Sayoud, 2017).

2. Waktu Kompresi dan Waktu Dekompresi

Proses kompresi membutuhkan waktu yang disebut sebagai waktu kompresi. Waktu dekompresi adalah waktu yang dibutuhkan dalam melakukan proses dekompresi.

2.6. Android

Sistem operasi berlandaskan *mobile* yang dikembangkan oleh persatuan pengembang perangkat lunak yang dikenal dengan nama *Open Handset Alliance* disebut dengan *Android*. Sistem operasi ini dapat didapatkan secara gratis dan bersifat *open source*. Biasanya *Android* sudah terinstal secara otomatis ketika membeli suatu ponsel pintar yang menggunakannya.

2.6.1. Aplikasi Berbasis Android

Aplikasi berbasis *Android* merupakan aplikasi yang dibuat untuk dijalankan pada sistem operasi *Android*. Pembuatan aplikasi untuk *Android* menggunakan seperangkat alat yang disebut *Android Software Development Kit (Android SDK)*. Java dan *Kotlin* merupakan bahasa pemrograman yang biasa digunakan dalam pembuatan aplikasi *Android*. Perangkat lunak yang dapat digunakan untuk memudahkan pembuatan aplikasi adalah *Android Studio*.

2.7. Penelitian yang Relevan

Beberapa studi sebelumnya yang terkait dengan proposal ini disajikan di bawah:

1. Pada penelitian yang berjudul “*Data Compression Using Stout Codes*” (Nasution, 2019) ditemukan bahwa penggunaan algoritma *Stout Code* untuk melakukan kompresi pada *file* teks berujung kepada keberhasilan dengan 60% *compression rate* dan 40 % *space saving*.
2. Pada penelitian yang berjudul “*Comparison Study of Fibonacci Code Algorithm and Even-Rodeh Algorithm for Data Compression*” (Rachmawati, et. al., 2019) didapatkan bahwa berdasarkan rasio kompresi, *Fibonacci Code* lebih baik dari *Even Rodeh Code* untuk karakter yang homogen dan heterogen dengan rasio rata-rata masing-masing adalah 0.25% dan 0.65%. Berdasarkan waktu dekompresi, *Even-Rodeh Code* lebih baik dari *Fibonacci Code* untuk karakter homogen dengan waktu rata-rata 0.004ms sementara untuk karakter heterogen, *Fibonacci Code* lebih baik dengan waktu rata-rata 0.279ms.
3. Pada penelitian yang berjudul “*Comparative Analysis Run-Length Encoding Algorithm and Fibonacci Code Algorithm on Image Compression*” (Hardi, et. al., 2019) ditemukan bahwa algoritma *Fibonacci Code* memberikan *compression ratio* dan *space saving* yang lebih baik dari *Run Length Encoding* pada citra berwarna sementara *Run Length Encoding* lebih baik pada citra *grayscale*.
4. Pada penelitian yang berjudul “*Data Compression using Fibonacci Sequence*” (Bhattacharyya, 2017) didapat bahwa pengkodean dengan algoritma *Shannon-Fano* dan *Huffman* menghasilkan *compression ratio*, *space saving*, dan *bitrate* yang baik. Tetapi untuk jumlah karakter yang lebih sedikit dan probabilitas yang lebih banyak, *Fibonacci Code* memberikan hasil yang lebih baik.

BAB 3

ANALISIS DAN PERANCANGAN

3.1. Analisis Sistem

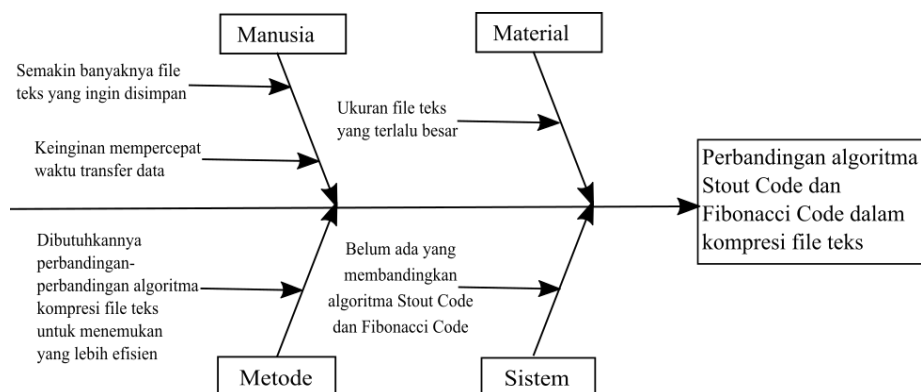
Analisis sistem merupakan tahapan mengidentifikasi hal-hal yang diperlukan untuk pembuatan sistem. Tujuannya adalah memudahkan dalam tahap perancangan sistem. Pada penelitian ini, analisis sistem terdiri dari dua aspek, yakni analisis permasalahan dan analisis kebutuhan.

3.1.1 Analisis Masalah

Analisis masalah merupakan tahapan mengidentifikasi faktor-faktor yang menyebabkan munculnya permasalahan. Ketika dilakukan analisis masalah, akan tampak jelas sebab-sebab dibuatnya sistem. Tujuannya adalah mendapatkan gambaran yang jelas terhadap solusi yang akan diterapkan ke dalam sistem.

Pada penelitian ini, permasalahan yang diangkat adalah bagaimana perbandingan algoritma *Stout Code* dan algoritma *Fibonacci Code* dalam melakukan kompresi *file* teks pada aplikasi berbasis *Android*. Adapun untuk memudahkan dalam mengenali faktor-faktor yang menyebabkan adanya permasalahan ini dan hubungan antara faktor-faktor tersebut, maka penulis memaparkannya ke dalam diagram *Ishikawa*.

Gambar 3.1 di bawah ini menunjukkan diagram *ishikawa* penelitian ini.



Gambar 3.1. Diagram *Ishikawa*

Gambar 3.1 menunjukkan bahwa permasalahan utama dari penelitian ini adalah membandingkan antara Algoritma *Stout Code* dan *Fibonacci Code* dalam kompresi *file* teks. Kemudian terdapat empat faktor penyebab masalah utama, yaitu manusia, material, metode dan sistem. Pada keempat faktor ini, terdapat rincian masalah yang berhubungan dengan masing-masing faktor digambarkan dengan arah panah yang mengarah ke panah milik faktor.

3.1.2 Analisis Kebutuhan

Setelah menyelesaikan tahap analisis masalah, maka didapatkan gambaran terhadap solusi yang akan diterapkan. Adapun untuk menerapkan solusi tersebut ke dalam suatu sistem, perlu dicari kebutuhan-kebutuhan yang harus ada pada sistem tersebut. Tujuannya adalah mendapatkan gambaran yang jelas terhadap sistem yang akan menerapkan solusi permasalahan. Tahapan dalam mengidentifikasi kebutuhan-kebutuhan sistem disebut dengan analisis kebutuhan.

Analisis kebutuhan yang harus dilakukan terdiri dari dua jenis, yakni kebutuhan fungsional dan kebutuhan non-fungsional. Kebutuhan fungsional adalah fungsi-fungsi yang harus terdapat dalam sistem agar sistem dapat dinyatakan efektif dalam menyelesaikan masalah. Kebutuhan non-fungsional mencakup elemen-elemen yang dibutuhkan dalam sistem agar sistem memenuhi standar kualitas yang baik.

3.1.2.1 Kebutuhan Fungsional

Berikut adalah daftar kebutuhan fungsional yang harus terpenuhi dalam sistem pada penelitian ini:

1. Sistem dapat mengenali *string* yang terdapat di dalam berkas dengan format *.txt*.
2. *File* teks dapat dikompres oleh sistem baik dengan algoritma *Stout Code* maupun dengan *Fibonacci Code*.
3. Dekompresi *file* yang berasal dari hasil kompresi dapat dilakukan oleh sistem baik menggunakan algoritma *Stout Code* maupun *Fibonacci Code*.
4. Sistem dapat menghitung rasio kompresi (*Compression ratio*) dan waktu kompresi/dekompresi.

3.1.2.2 Kebutuhan Non-Fungsional

Berikut adalah daftar kebutuhan non-fungsional yang diharapkan terpenuhi dalam sistem pada penelitian ini:

1. Portabilitas

Sistem dapat beroperasi pada setiap *smartphone* yang menggunakan versi 5.0 sampai terbaru dari sistem operasi *Android*.

2. Ketersediaan

Sistem dapat dijalankan kapan saja.

3. Ergonomi

Sistem memiliki antarmuka yang sederhana dengan fungsi yang minimalis sehingga tidak membingungkan pengguna.

4. Keamanan

Sistem tidak memiliki hal-hal yang membahayakan perangkat pengguna.

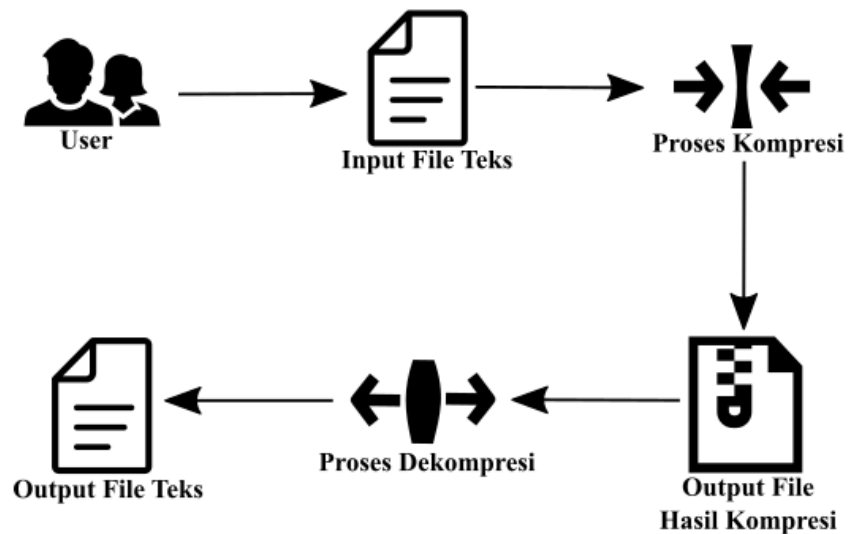
3.2. Pemodelan Sistem

Pemodelan sistem adalah tahapan membuat bentuk sederhana dari sebuah sistem yang direpresentasikan secara visual dalam bentuk diagram-diagram. Penelitian ini melakukan pemodelan sistem dengan menggunakan beberapa diagram, yaitu *use case* diagram, *activity* diagram, dan *sequence* diagram.

3.2.1 Arsitektur Umum

Diagram arsitektur umum digunakan untuk merepresentasikan secara keseluruhan proses yang terjadi pada sistem.

Gambar 3.2 di bawah ini merupakan arsitektur umum dari sistem



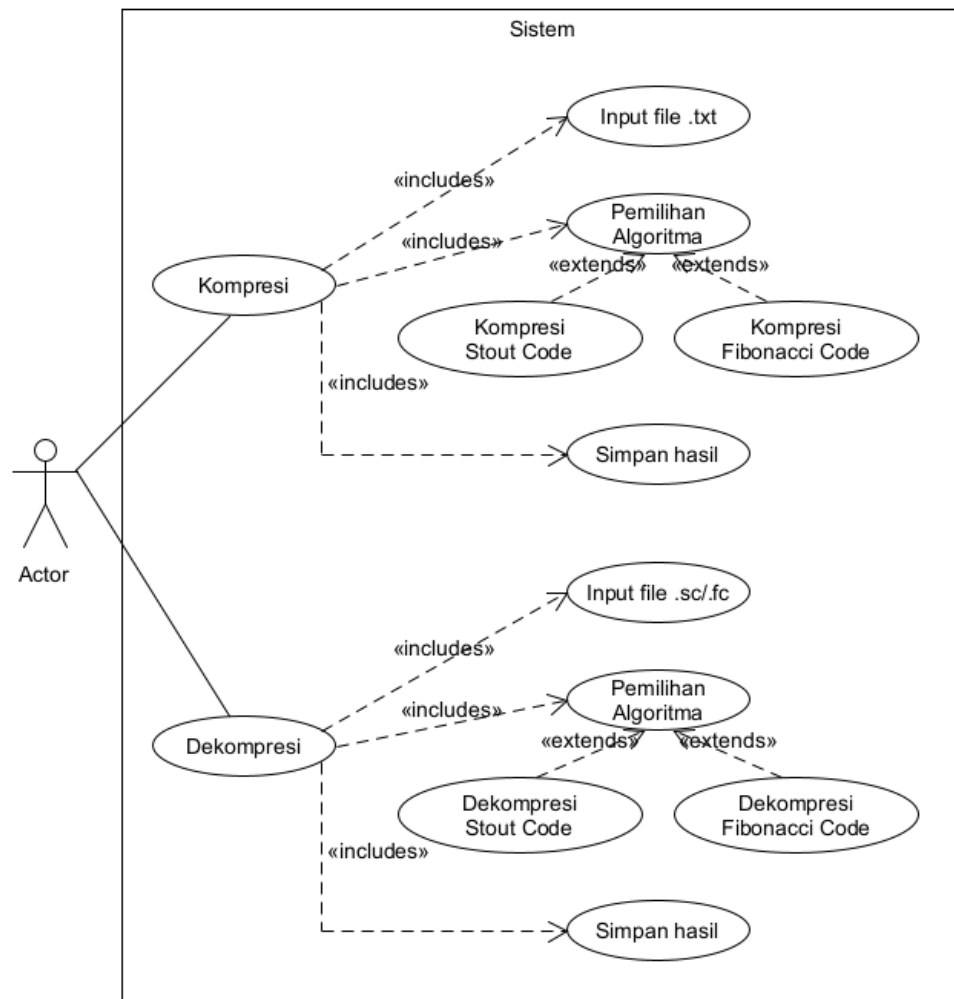
Gambar 3.2. Arsitektur Umum

Pada gambar tersebut pengguna (*user*) memasukkan *file* teks ke dalam sistem. *File* tersebut dikompresi dengan menggunakan algoritma *Stout Code* ataupun *Fibonacci Code*. Kemudian sistem mengeluarkan *file* hasil kompresi. Untuk mengembalikan ke *file* aslinya, *file* tersebut didekompresi terlebih dahulu baru kemudian dapat dikeluarkan.

3.2.2 Use case Diagram

Diagram ini digunakan dalam menggambarkan fungsi-fungsi yang terdapat dalam sebuah sistem, hubungan antar fungsi, dan hubungan antara sistem dengan pihak luar (aktor).

Untuk melihat *use case* diagram dari sistem yang akan dikembangkan pada penelitian ini, dapat dilihat pada gambar 3.3.



Gambar 3.3. Use case Diagram

Gambar tersebut menunjukkan bahwa pengguna merupakan satu-satunya aktor yang berinteraksi dengan sistem. Kemudian terdapat dua fungsi utama pada sistem, yaitu fungsi kompresi dan dekompresi. Ketika sistem dijalankan, pengguna dapat langsung memilih antara kedua fungsi ini.

Fungsi kompresi melibatkan fungsi untuk memasukkan *file*, memilih algoritma dan menyimpan hasil. Dalam melakukan fungsi pemilihan algoritma, pengguna wajib memilih antara algoritma *Stout Code* dan *Fibonacci Code*. Fungsi dekompresi melibatkan fungsi untuk memasukkan *file*, memilih algoritma dan menyimpan hasil. *File* yang dimasukkan dapat berekstensi **.sc* atau **.fc*.

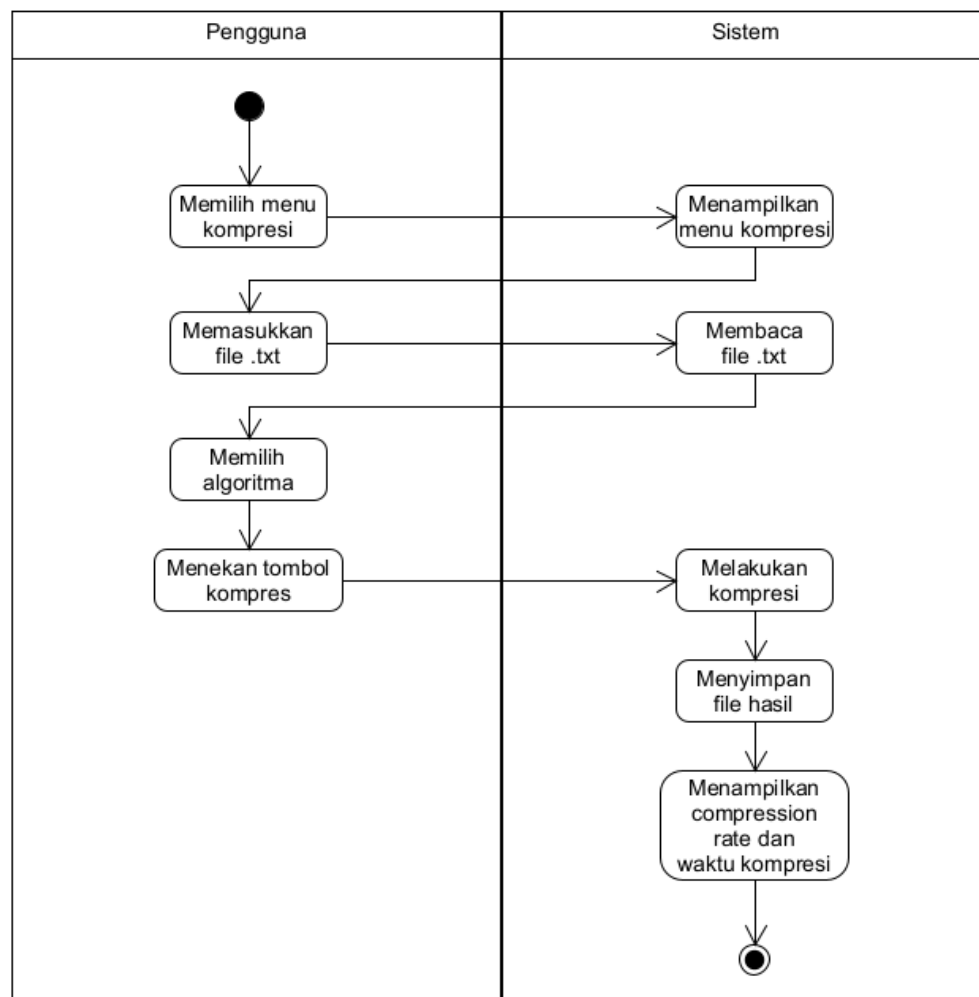
3.2.3 Activity Diagram

Activity diagram adalah suatu bentuk visualisasi dari aktivitas-aktivitas yang terjadi pada sistem dan bagaimana mereka dijalankan.

3.2.3.1 Activity Diagram Proses Kompresi

Diagram berikut ini menggambarkan aktivitas yang terlibat dalam proses kompresi.

Gambar 3.4 di bawah ini merupakan *activity* diagram kompresi.



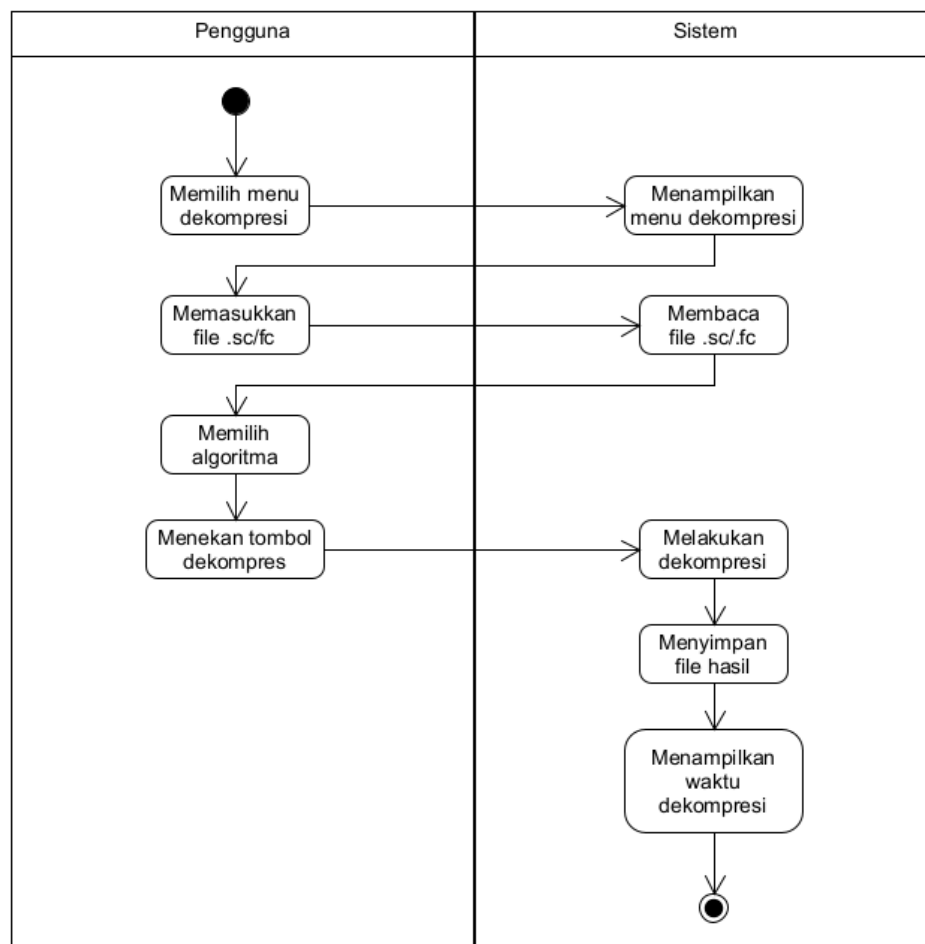
Gambar 3.4. Activity Diagram Proses Kompresi

Aktivitas diawali dengan pengguna memilih opsi kompresi pada menu. Kemudian menu kompresi akan tampil. Berikutnya, pengguna mencari dan memilih *file* teks yang kemudian dibaca oleh sistem, Pengguna selanjutnya menetapkan algoritma apa yang akan digunakan untuk melakukan proses kompresi. Lalu pengguna menekan tombol kompres. Sistem kemudian melakukan kompresi dan apabila selesai, menyimpan *file* yang sudah dikompresi serta menampilkan nilai *compression ratio* dan waktu kompresi.

3.2.3.2 Activity Diagram Proses Dekompresi

Diagram berikut menggambarkan aktivitas yang terlibat dalam proses dekompresi.

Gambar 3.5 di bawah ini merupakan *activity* diagram dari proses dekompresi.



Gambar 3.5. Activity Diagram Proses Dekompresi

Aktivitas diawali dengan pengguna memilih opsi dekompresi pada menu. Kemudian menu dekompresi akan tampil. Berikutnya, pengguna mencari dan memilih *file* kompresan yang kemudian dibaca oleh sistem, Pengguna selanjutnya menetapkan algoritma apa yang sesuai untuk melakukan proses dekompresi. Lalu pengguna menekan tombol dekompres. Sistem kemudian melakukan dekompresi dan apabila selesai, menyimpan *file* asli dan menampilkan waktu dekompresi.

3.2.4 Sequence Diagram

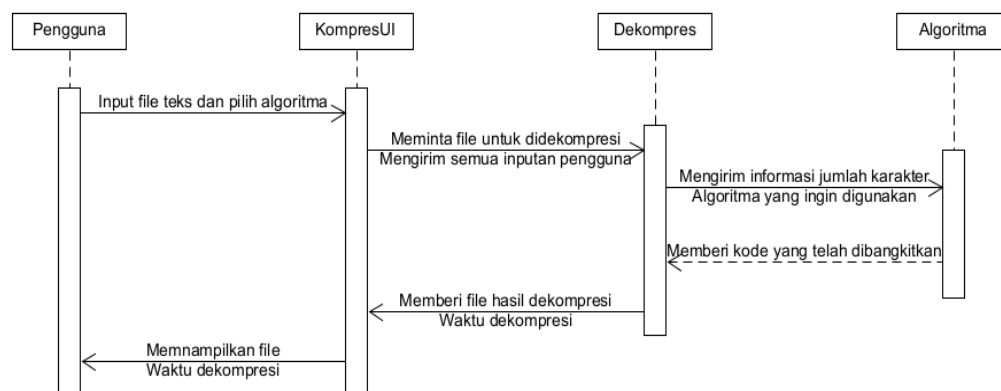
Sequence Diagram adalah diagram yang menggambarkan bagaimana objek-objek di dalam sistem berinteraksi yang disusun berdasarkan urutan waktu. Objek dilambangkan dengan persegi panjang berisi nama objek. Interaksi antar objek dilambangkan dengan arah panah yang disebut dengan pesan.

Pada penelitian ini, *sequence diagram* dibagi menjadi dua bagian yaitu *sequence diagram* proses kompresi dan *sequence diagram* proses dekompresi.

3.2.4.1 Sequence Diagram Proses Kompresi

Berikut ini adalah *sequence diagram* untuk proses kompresi.

Gambar 3.6 di bawah ini merupakan *sequence diagram* dari proses kompresi.



Gambar 3.6. Sequence Diagram Proses Kompresi

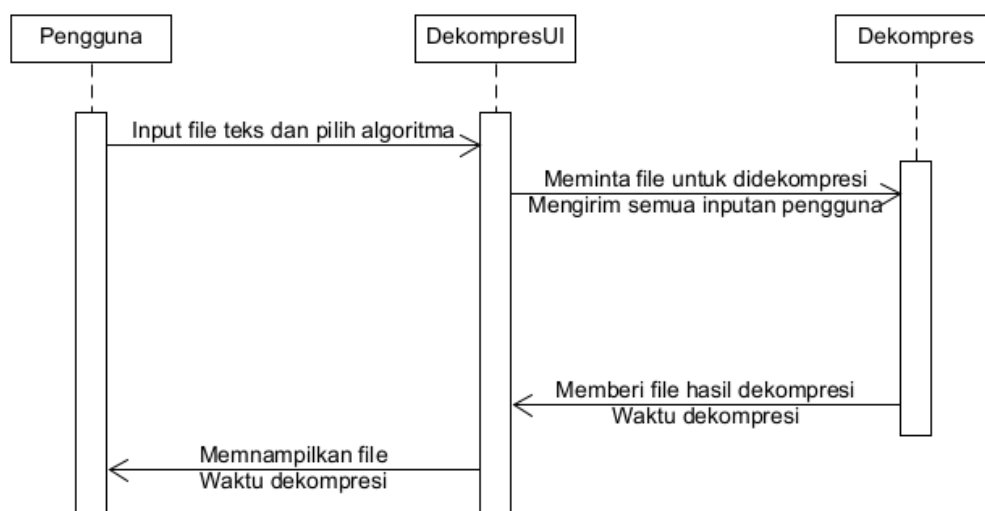
Pengguna memasukkan *file* teks dan memilih algoritma yang ingin digunakan untuk melakukan kompresi pada *form* yang telah disediakan. *Form* tersebut terletak pada objek dengan kelas *KompresUI* yang merupakan penyedia antarmuka menu kompresi. Kemudian informasi tersebut dikirim ke objek kompres untuk dilakukan proses kompresi.

Pada objek kompres, jumlah karakter pada *file* teks dihitung dan dikirim ke objek algoritma bersama informasi algoritma yang ingin digunakan untuk melakukan kompresi. Sesuai algoritma, kode dibangkitkan dan dikirim ke objek kompres. Kemudian objek kompres membuat dan menyimpan tabel *encoding*. *File* teks dikompresi sesuai dengan tabel encoding. Nilai *compression ratio* dan waktu kompresi dihitung jumlahnya dan ditampilkan kepada pengguna.

3.2.4.2 Sequence Diagram Proses Dekompresi

Berikut ini adalah *sequence* diagram untuk proses dekompresi.

Gambar 3.7 di bawah ini merupakan *sequence* diagram dari proses dekompresi.



Gambar 3.7. Sequence Diagram Proses Dekompresi

Pengguna memasukkan *file* teks dan memilih algoritma yang ingin digunakan untuk melakukan dekompresi. *Form* tersebut ditampilkan melalui objek dengan kelas *DekompresUI*. Kemudian informasi tersebut dikirim ke objek dekompres untuk dilakukan proses dekompresi. Sesuai dengan tabel *encoding* yang ada, *file* teks didekompresi dan dihitung waktu dekompresinya. Hasil tersebut dikembalikan ke objek *DekompresUI* untuk ditampilkan kepada pengguna.

3.3. Flowchart

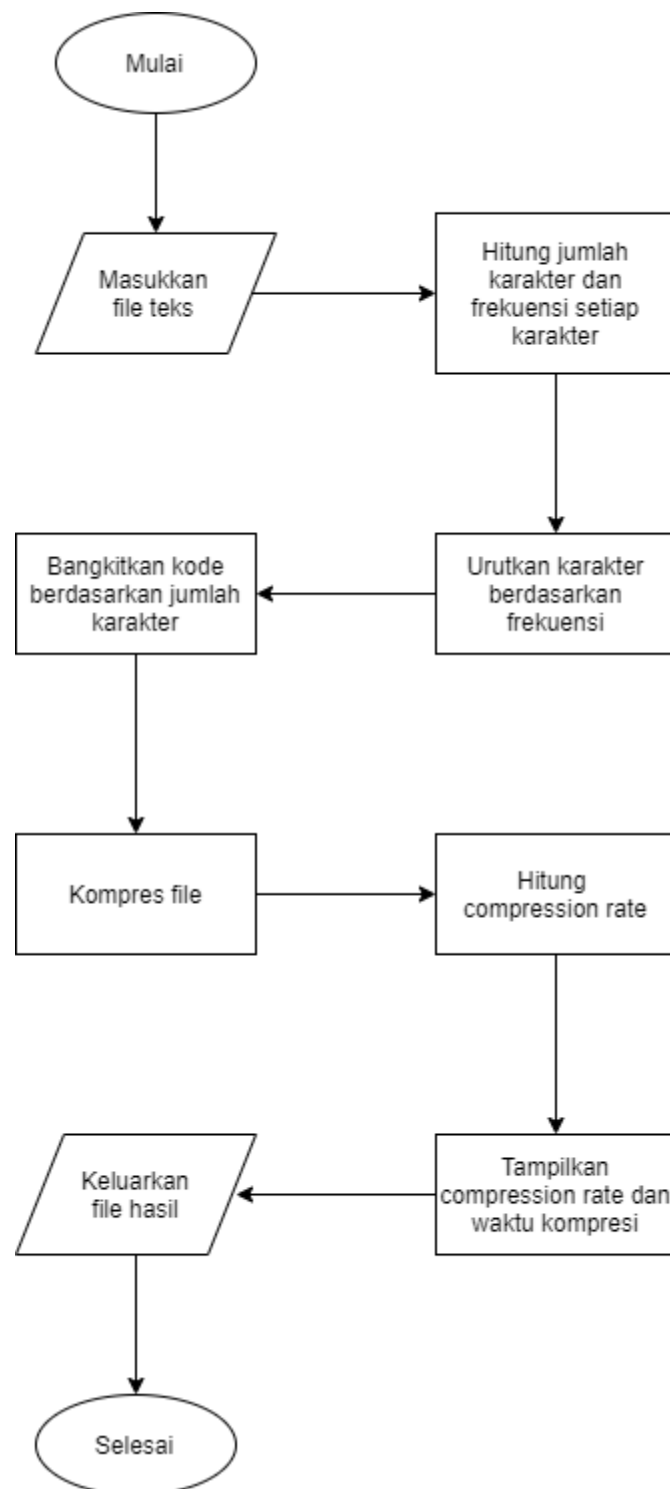
Flowchart merupakan diagram yang menunjukkan alir kerja atau algoritma dalam menyelesaikan masalah.

Pada penelitian ini terdapat beberapa *flowchart*, yaitu *flowchart* sistem, *flowchart* pembangkitan *Stout Code*, dan *flowchart* pembangkitan *Fibonacci Code*.

3.3.1 Flowchart Sistem

Berikut ini adalah *flowchart* proses kompresi pada sistem secara keseluruhan.

Gambar 3.8 di bawah ini merupakan *flowchart* proses kompresi.

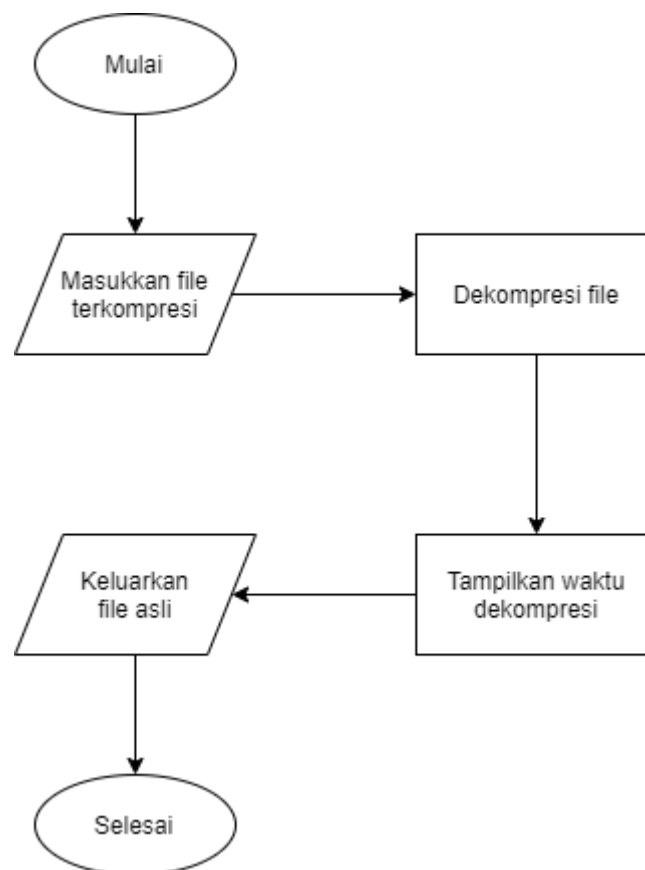


Gambar 3.8. Flowchart Proses Kompresi

Proses kompresi dimulai dengan pengguna memasukkan *file* teks. Kemudian sistem menghitung jumlah karakter beserta frekuensi setiap karakter dan mengurutkan karakter sesuai dengan frekuensi. Sesuai dengan algoritma yang telah dipilih, kode dibangkitkan sebanyak jumlah karakter yang ada. Selanjutnya *file* teks dikompresi dengan mengganti setiap karakter pada *file* sesuai dengan kode bersangkutan yang telah dibangkitkan. Selesai dikompresi, compression rate dihitung dan ditampilkan di *User Interface*. *File* hasil kompresi dapat disimpan oleh pengguna.

Berikut ini adalah *flowchart* proses dekomposisi pada sistem secara keseluruhan.

Gambar 3.9 di bawah ini merupakan *flowchart* proses dekomposisi.



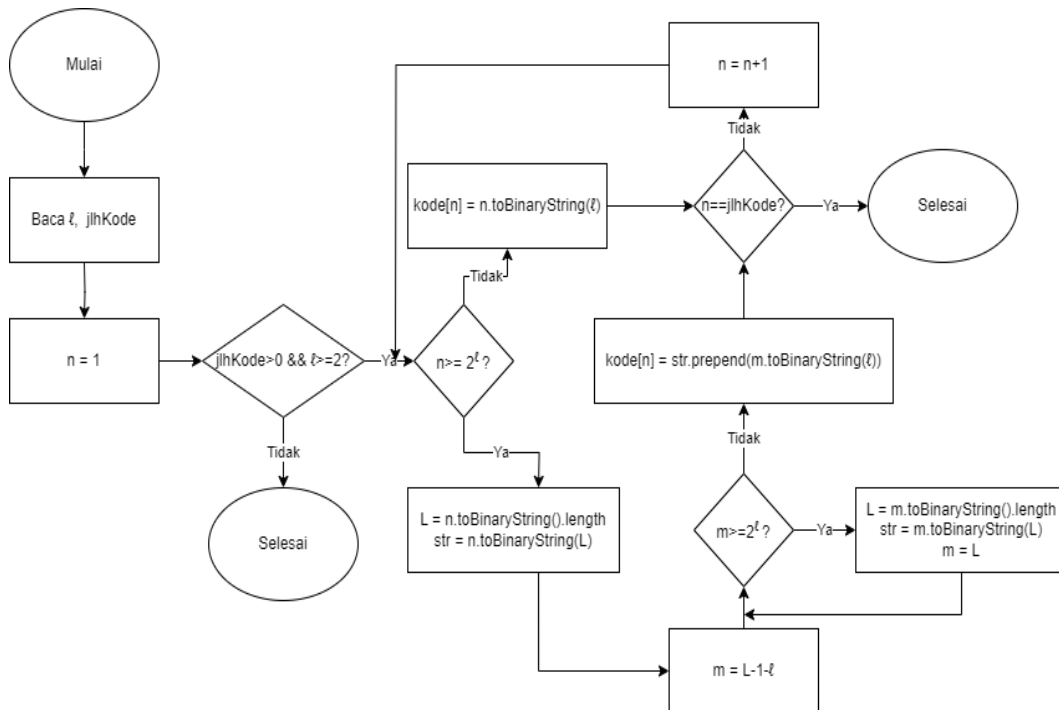
Gambar 3.9. Flowchart Proses Dekomposisi

Proses dekomposisi dimulai dengan pengguna memasukkan *file* terkompresi. Sesuai dengan algoritma yang telah dipilih, sistem melakukan dekomposisi *file*. Selanjutnya waktu dekomposisi ditampilkan di *user interface*. *File* hasil dekomposisi dapat disimpan oleh pengguna.

3.3.2 Flowchart Stout Code

Berikut ini adalah *flowchart* pembangkitan kode dengan algoritma *Stout Code*. Algoritma *Stout Code* yang digunakan adalah keluarga S_ℓ .

Gambar 3.10 di bawah ini merupakan *flowchart* pembangkitan *Stout Code*.



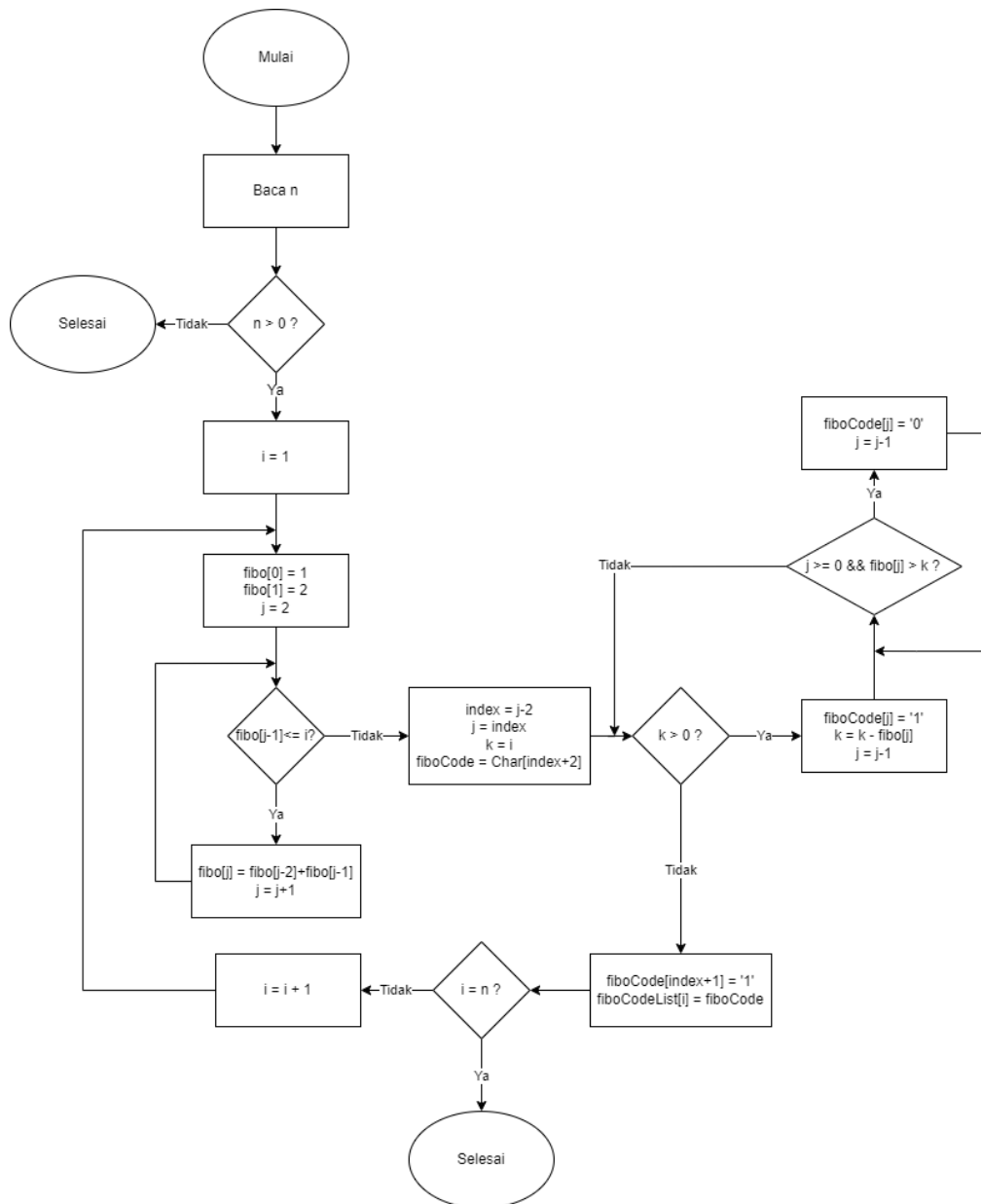
Gambar 3.10. Flowchart Pembangkitan Stout Code

Sistem membaca parameter ' ℓ ' dan jumlah kode yang diinginkan. Kemudian dibutuhkan variabel bantu ' n ' untuk memberhentikan pembangkitan kode apabila kode yang dibangkitkan sudah sebanyak jumlah kode. Sistem memastikan bahwa jumlah kode yang dibutuhkan lebih dari nol dan ℓ lebih besar atau sama dengan dua. Selanjutnya rumus *Stout Code* keluarga S_ℓ dijalankan. Apabila n lebih besar atau sama dengan 2^ℓ , maka kode dibangkitkan dengan rumus $R_\ell(L - 1 - \ell)B(n, \ell)$ sedangkan jika tidak maka kode dibangkitkan hanya dengan $B(n, \ell)$ saja. Pada gambar, fungsi $\text{toBinaryString}(\ell)$ merupakan representasi dari $B(n, \ell)$ yaitu mengubah nilai n ke nilai biner dengan digit sebanyak ℓ . Apabila fungsi digunakan tanpa parameter, maka digit tidak ditentukan. Fungsi $\text{prepend}()$ digunakan untuk menyisipkan suatu *string* ke indeks pertama dari *string* yang lain. Hasil kode disimpan dalam sebuah variabel *array* yang pada gambar direpresentasikan dengan variabel '*kode*'.

3.3.3 Flowchart Fibonacci Code

Berikut ini adalah *flowchart* pembangkitan kode sebanyak n buah dengan algoritma *Fibonacci Code*.

Gambar 3.11 di bawah ini merupakan *flowchart* pembangkitan *Fibonacci Code*.



Gambar 3.11. Flowchart Pembangkitan Fibonacci Code

Sebelum proses dimulai, sistem memastikan nilai n lebih besar dari nol. Langkah pertama adalah mencari bilangan Fibonacci yang besarnya lebih kecil dan mendekati nilai n . Variabel 'i' digunakan sebagai pengganti nilai 'n' dalam proses perulangan. Setelah nilai Fibonacci didapat, maka langkah berikutnya adalah membangkitkan kode digit per digit dan memasukkannya ke dalam variabel *array* bertipe data *char*. Pada gambar variabel ini direpresentasikan dengan 'fiboCode'. Kode yang telah lengkap dimasukkan ke variabel *array* lain yang pada gambar direpresentasikan dengan 'fiboCodeList'. Langkah terakhir adalah memeriksa apakah nilai 'i' sudah sama dengan 'n'. Jika sudah, berhenti membangkitkan kode. Jika belum, bangkitkan kode selanjutnya.

3.4 Perancangan Antarmuka

Antarmuka pada sistem ini akan dibangun dengan bahasa XML. Sebelum membuat antarmuka, ada baiknya dirancang terlebih dahulu menggunakan sketsa. Adapun pada sistem ini, halaman antarmuka terbagi atas dua yaitu, halaman kompresi dan dekompresi.

Berikut adalah sketsa halaman kompresi.

Gambar 3.12 di bawah ini merupakan sketsa untuk halaman kompresi.

Sketsa halaman kompresi yang menunjukkan layout elemen-elemen antarmuka:

- 1. Input field untuk file yang akan dikompresi.
- 2. Tombol 'Browse' untuk memilih file.
- 3. Area preview atau informasi tambahan.
- 4. Input field untuk 'Besar File'.
- 5. Pilihan algoritma dengan dua opsi: 'Stout Code' dan 'Fibonacci Code'.
- 6. Tombol 'KOMPRES FILE'.
- 7. Output field untuk 'Besar Hasil Kompresi'.
- 8. Output field untuk 'Rasio Kompresi'.
- 9. Output field untuk 'Waktu Kompresi'.
- 10. Tombol 'Kompresi' dan 'Dekompresi'.

Gambar 3.12. Sketsa Halaman Kompresi

Tabel 3.1. Keterangan Sketsa Halaman Kompresi

No	Keterangan
1	TextView yang berisikan path dari <i>file</i> teks yang akan dikompresi
2	Button untuk melakukan pencarian <i>file</i> pada penyimpanan <i>Android</i>
3	TextView untuk menampilkan isi <i>file</i> teks yang telah dipilih
4	TextView yang menampilkan ukuran <i>file</i> teks
5	RadioGroup yang berisikan dua buah RadioButton untuk memilih algoritma yang akan dipakai untuk melakukan kompresi
6	Button untuk menjalankan proses kompresi
7	TextView yang menampilkan besar hasil kompresi setelah <i>file</i> berhasil dikompresi
8	TextView yang menampilkan rasio kompresi
9	TextView yang menampilkan waktu yang dibutuhkan saat melakukan kompresi
10	BottomNavigationView yang digunakan untuk berpindah halaman

Tabel 3.1 di atas menunjukkan keterangan bagian-bagian yang terdapat di dalam sketsa halaman kompresi.

Berikut adalah sketsa halaman Dekompresi.

Gambar 3.13 di bawah ini merupakan sketsa dari halaman dekompresi.

1	2 Browse
3	
Besar File Terkompresi : 4	
Pilih Algoritma Dekompresi 5 <input type="radio"/> Stout Code <input type="radio"/> Fibonacci Code	
6 DEKOMPRES FILE	
Besar Hasil Dekompresi: 7	
Waktu Dekompresi: 8	
Kompresi	9 Dekompresi

Gambar 3.13. Sketsa Halaman Dekompresi

Tabel 3.2. Keterangan Sketsa Halaman Dekompresi

No	Keterangan
1	TextView yang berisikan path dari <i>file</i> teks yang akan didekompresi
2	Button untuk melakukan pencarian <i>file</i> yang akan didekompresi pada penyimpanan <i>Android</i>
3	TextView untuk menampilkan isi <i>file</i> yang telah dipilih
4	TextView yang menampilkan ukuran <i>file</i>
5	RadioGroup yang berisikan dua buah RadioButton untuk memilih algoritma yang akan dipakai untuk melakukan dekomposisi
6	Button untuk menjalankan proses dekomposisi
7	TextView yang menampilkan besar <i>file</i> setelah berhasil didekompresi (besar semula)
8	TextView yang menampilkan waktu yang dibutuhkan saat melakukan dekomposisi
9	BottomNavigationView yang digunakan untuk berpindah halaman

Tabel 3.2 di atas menunjukkan keterangan bagian-bagian yang terdapat di dalam sketsa halaman dekomposisi.

BAB 4

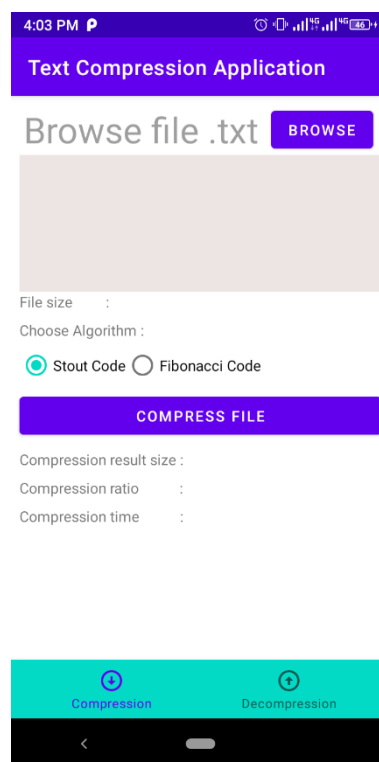
IMPLEMENTASI DAN PENGUJIAN SISTEM

4.1 Implementasi Sistem

Pada tahapan ini, sistem dibangun sesuai dengan analisis dan rancangan yang telah dilakukan sebelumnya. Bahasa pemrograman yang digunakan untuk membangun sistem adalah *Kotlin*. Aplikasi yang digunakan untuk membantu pemrograman adalah *Android Studio*. Dua buah *Fragment* digunakan pada sistem, yaitu *Fragment Kompresi* dan *Fragment Dekompresi*.

4.1.1 *Fragment Kompresi*

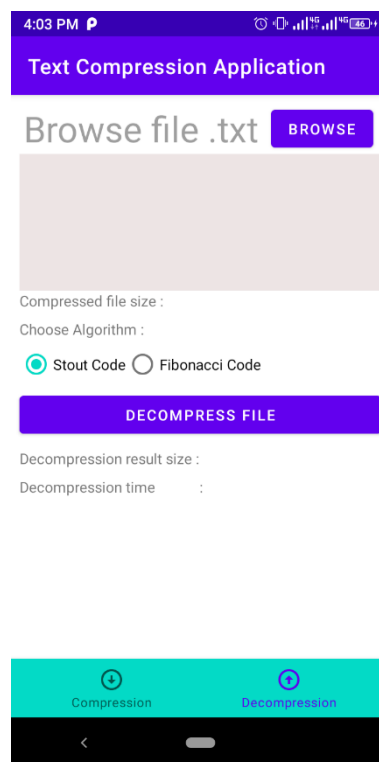
Fragment kompresi merupakan halaman pertama yang tampil saat sistem dijalankan, di *fragment* ini terdapat komponen – komponen untuk memulai proses kompresi pada *file* teks. Hasil *fragment* kompresi dapat dilihat pada Gambar 4.1 berikut.



Gambar 4.1. *Fragment* Kompresi

4.1.2 Fragment Dekompresi

Fragment dekomposisi berisikan komponen – komponen untuk melakukan proses dekomposisi. Pada *fragment* ini terdapat pilihan metode untuk pengguna dimana yang berguna untuk algoritma mana yang akan di dekomposisi terlebih dahulu. Hasil *fragment* dekomposisi dapat dilihat pada Gambar 4.2 berikut.



Gambar 4.2. *Fragment* Dekompresi

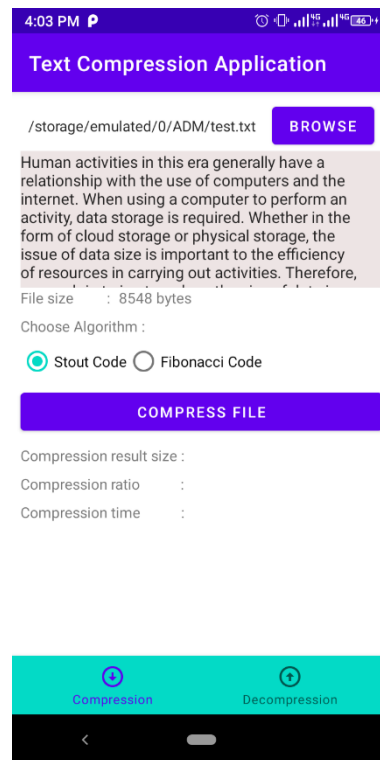
4.2 Pengujian Sistem

Tahap ini bertujuan untuk mengevaluasi hasil implementasi sistem agar sesuai dengan fungsi-fungsi yang telah ditentukan pada tahap analisis dan perancangan sistem. Pengujian sistem dilakukan pada *file* teks yang berformat *.txt. Dalam penelitian ini, pengujian sistem dibagi menjadi dua proses utama, yaitu pengujian pada proses kompresi dan proses dekomposisi.

4.2.1 Pengujian Proses Kompresi

Pada tahap pengujian proses kompresi, hal pertama yang dilakukan pengguna adalah memasukkan *file* teks. Selanjutnya, pengguna dapat memilih algoritma mana yang diinginkan terlebih dulu untuk melakukan proses kompresi.

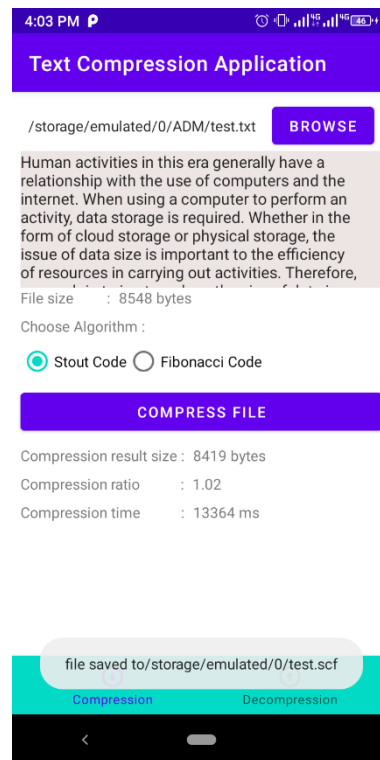
Gambar 4.3 di bawah ini merupakan tampilan saat pengujian kompresi.



Gambar 4.3. Pengujian Kompresi

Setelah itu pengguna menekan tombol kompres untuk menjalankan sistem tersebut. Kemudian *file* tersebut akan tersimpan pada folder *root* dari penyimpanan internal *android* pengguna. Selanjutnya sistem akan menghasilkan perhitungan besar hasil kompresi, rasio kompresi dan waktu kompresi.

Gambar 4.4 di bawah ini menunjukkan tampilan kompresi berhasil.

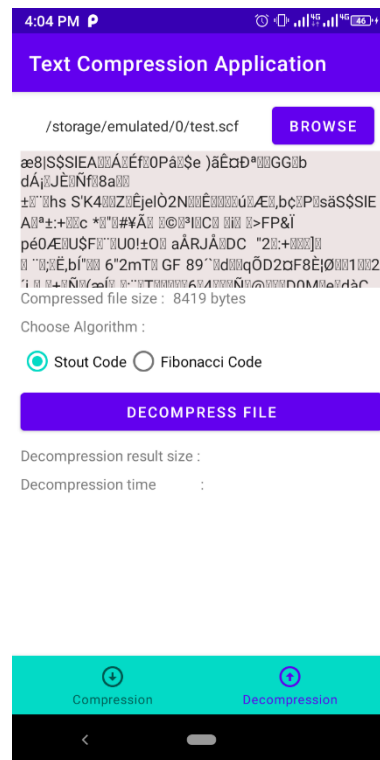


Gambar 4.4. Kompresi Berhasil

4.2.2 Pengujian Proses Dekompresi

Langkah awal yang dilakukan adalah memasukkan berkas teks yang sudah dikompres. Selanjutnya, pengguna dapat memilih algoritma mana yang sesuai dengan ekstensi berkas terkompresi agar proses dekompresi dapat berjalan dengan benar.

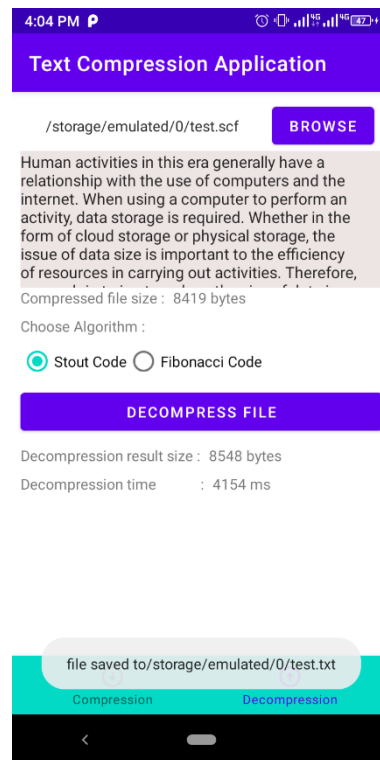
Gambar 4.5 di bawah ini merupakan tampilan saat pengujian dekompresi.



Gambar 4.5. Pengujian Dekompresi

Setelah itu pengguna menekan tombol dekompres untuk menjalankan proses dekompresi tersebut. Kemudian *file* tersebut akan tersimpan pada folder *root* dari penyimpanan internal *android* pengguna. Sistem akan menghasilkan perhitungan besar hasil dekompresi dan waktu dekompresi.

Gambar 4.6 di bawah ini menunjukkan tampilan dekompresi berhasil.



Gambar 4.6. Dekompresi Berhasil

4.2.3 Hasil Pengujian

Pengujian dilangsungkan terhadap dua jenis *string*, yakni *string* homogen (satu jenis karakter) dan *string* heterogen (jenis karakter yang beberapa macam). Sampel kedua macam *string* dapat dilihat pada kedua tabel berikut.

Tabel 4.1. Sampel Pengujian *String* Homogen

Frekuensi Karakter	<i>String</i> Homogen
500	ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ...Z
1000	ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ...Z
2000	ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ...Z
3000	ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ...Z
4000	ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ...Z
5000	ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ...Z
10000	ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ...Z

Tabel 4.1 di atas menunjukkan isi sampel untuk pengujian *string* homogen.

Tabel 4.2. Sampel Pengujian *String* Heterogen

Frekuensi Karakter	Jenis Karakter	<i>String</i> Heterogen
500	10	QWERT67890QWERT67890QWERT67890QWE RT67890...
1000	10	QWERT67890QWERT67890QWERT67890QWE RT67890...
2000	10	QWERT67890QWERT67890QWERT67890QWE RT67890...
3000	10	QWERT67890QWERT67890QWERT67890QWE RT67890...
4000	10	QWERT67890QWERT67890QWERT67890QWE RT67890...
5000	10	QWERT67890QWERT67890QWERT67890QWE RT67890...
10000	10	QWERT67890QWERT67890QWERT67890QWE RT67890...

Tabel 4.2 di atas menunjukkan isi sampel untuk pengujian *string* heterogen.

4.2.4 Pengujian String Homogen

Dalam tabel 4.3 dan tabel 4.4 berikut, terdapat hasil uji *String* Homogen yang dilakukan oleh algoritma *Stout Code* dan algoritma *Fibonacci Code*.

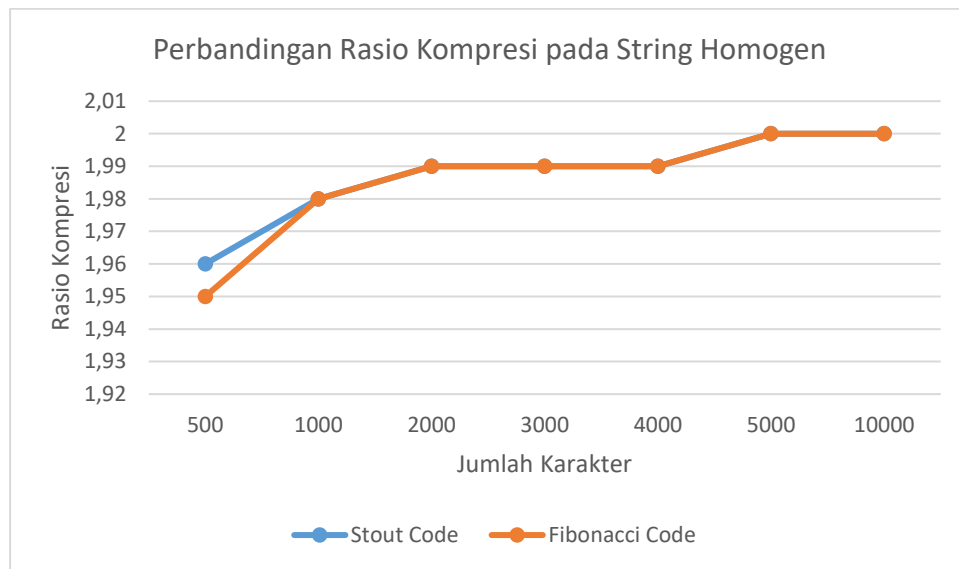
Tabel 4.3. Hasil Pengujian *String* Homogen dengan *Stout Code*

<i>Stout Code</i>					
Jumlah Karakter	Besar sebelum kompresi (bytes)	Besar sesudah kompresi (bytes)	Rasio Kompresi	Waktu Kompresi (ms)	Waktu Dekompresi (ms)
500	500	255	1,96	3	60
1000	1000	506	1,98	4	71
2000	2000	1006	1,99	7	177
3000	3000	1506	1,99	10	233
4000	4000	2006	1,99	20	313
5000	5000	2506	2	23	335
10000	10000	5006	2	36	1056
Rata - rata			1,99	14,71	320,71

Tabel 4.4. Hasil Pengujian *String* Homogen dengan *Fibonacci Code*

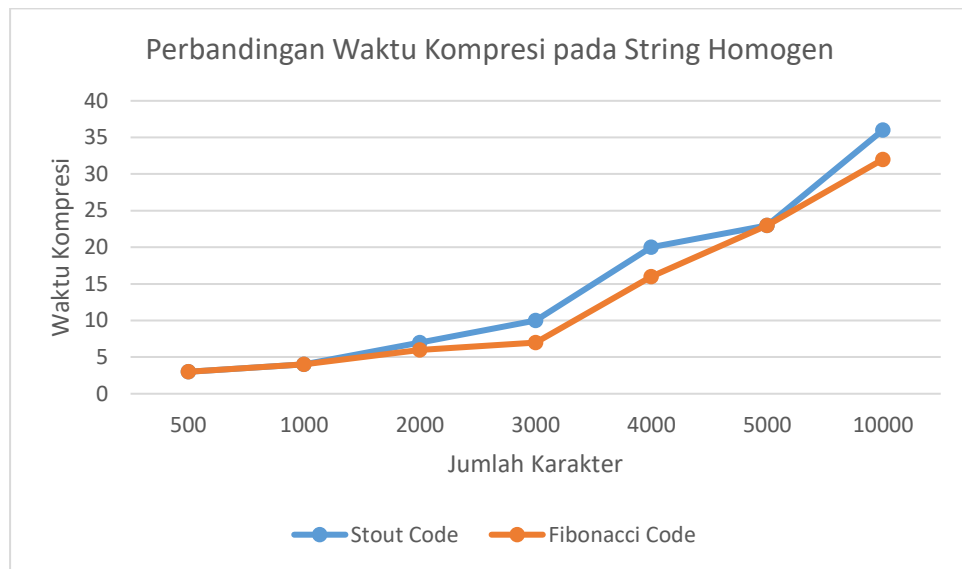
<i>Fibonacci Code</i>					
Jumlah Karakter	Besar sebelum kompresi (bytes)	Besar sesudah kompresi (bytes)	Rasio Kompresi	Waktu Kompresi (ms)	Waktu Dekompresi (ms)
500	500	256	1,95	3	10
1000	1000	506	1,98	4	12
2000	2000	1006	1,99	6	24
3000	3000	1506	1,99	7	33
4000	4000	2006	1,99	16	37
5000	5000	2506	2	23	50
10000	10000	5006	2	32	115
Rata - rata			1,99	13	40,14

Setelah hasil diperoleh dari pengujian yang ditampilkan pada Tabel 4.3 dan Tabel 4.4, maka dapat dibuat grafik perbandingan hasil pengujian algoritma *Stout Code* dan algoritma *Fibonacci Code* untuk *string* homogen berdasarkan rasio kompresi, waktu kompresi, dan waktu dekompresi. Gambar 4.7, Gambar 4.8 dan Gambar 4.9 menampilkan ketiga grafik tersebut.



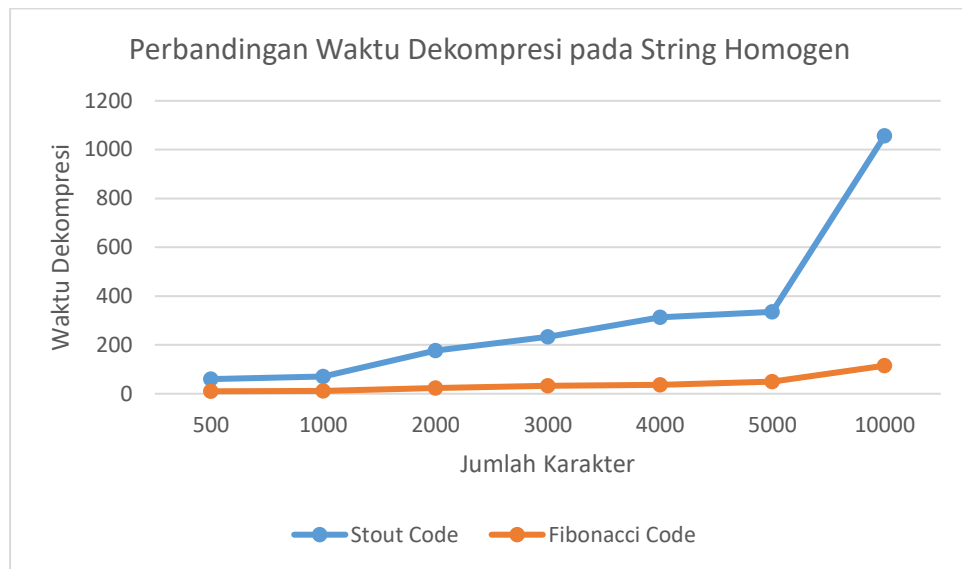
Gambar 4.7. Diagram Perbandingan Rasio Kompresi pada *String* Homogen

Dari Gambar 4.7, dapat dilihat bahwa algoritma *Stout Code* memiliki nilai rasio kompresi yang sedikit lebih besar dari *Fibonacci Code*. Artinya ukuran *file* hasil kompresi menggunakan algoritma *Stout Code* secara umum sedikit lebih kecil daripada menggunakan algoritma *Fibonacci Code*. Akan tetapi, karena perbandingan keduanya tidak begitu signifikan maka dapat disimpulkan bahwa perbandingan rasio kompresi algoritma *Stout Code* dan *Fibonacci Code* pada *String* Homogen adalah serupa.



Gambar 4.8. Diagram Perbandingan Waktu Kompresi pada *String* Homogen

Dari Gambar 4.8, dapat dilihat bahwa algoritma *Fibonacci Code* memerlukan waktu yang sedikit lebih cepat untuk melakukan proses kompresi daripada algoritma *Stout Code*. Akan tetapi perbedaan waktu kompresi tidak begitu besar. Oleh karena itu, dapat disimpulkan bahwa waktu yang diperlukan algoritma *Stout Code* dan *Fibonacci Code* untuk mengkompresi *file String* Homogen adalah serupa.



Gambar 4.9. Diagram Perbandingan Waktu Dekompresi pada *String* Homogen

Dari Gambar 4.9, dapat dilihat bahwa algoritma *Fibonacci Code* memerlukan waktu lebih cepat untuk melakukan proses dekomposisi daripada algoritma *Stout Code*. Dengan perbedaan waktu yang cukup signifikan, dapat disimpulkan bahwa algoritma *Fibonacci Code* lebih cepat daripada algoritma *Stout Code* dalam melakukan dekomposisi *file String* Homogen.

4.2.5 Pengujian String Heterogen

Dalam tabel 4.3 dan tabel 4.4 berikut, terdapat hasil uji *String* Heterogen yang dilakukan oleh algoritma *Stout Code* dan algoritma *Fibonacci Code*.

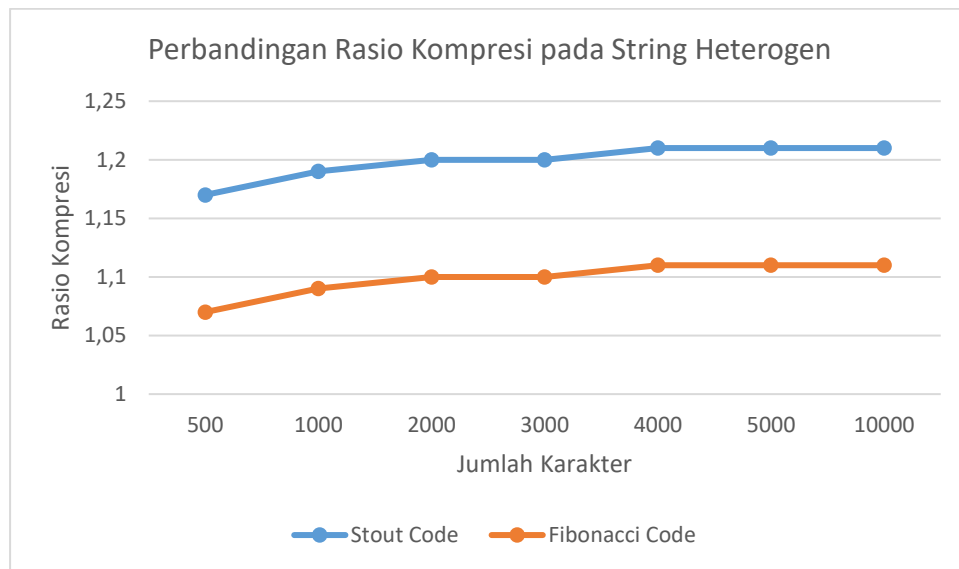
Tabel 4.5. Hasil Pengujian *String* Heterogen dengan *Stout Code*

<i>Stout Code</i>					
Jumlah Karakter	Besar sebelum kompresi (bytes)	Besar sesudah kompresi (bytes)	Rasio Kompresi	Waktu Kompresi (ms)	Waktu Dekompresi (ms)
500	500	428	1,17	6	102
1000	1000	840	1,19	10	162
2000	2000	1665	1,2	12	226
3000	3000	2490	1,2	23	437
4000	4000	3315	1,21	26	555
5000	5000	4140	1,21	27	674
10000	10000	8265	1,21	71	1466
Rata - rata			1,2	25	517,43

Tabel 4.6. Hasil Pengujian *String* Heterogen dengan *Fibonacci Code*

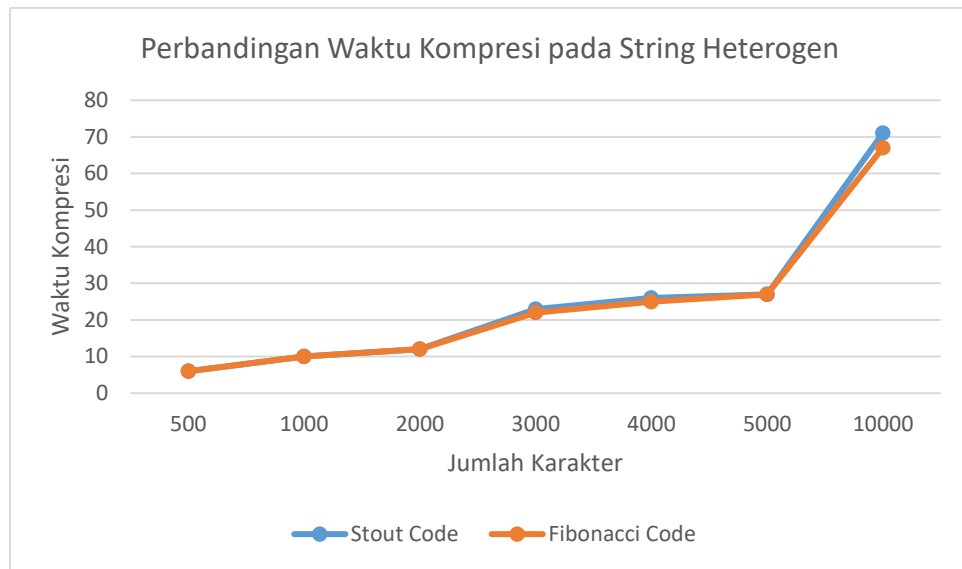
<i>Fibonacci Code</i>					
Jumlah Karakter	Besar sebelum kompresi (bytes)	Besar sesudah kompresi (bytes)	Rasio Kompresi	Waktu Kompresi (ms)	Waktu Dekompresi (ms)
500	500	466	1,07	6	17
1000	1000	915	1,09	10	31
2000	2000	1815	1,1	12	70
3000	3000	2715	1,1	22	100
4000	4000	3615	1,11	25	127
5000	5000	4515	1,11	27	157
10000	10000	9015	1,11	67	203
Rata - rata			1,1	24,14	100,71

Setelah hasil diperoleh dari pengujian yang ditampilkan pada Tabel 4.5 dan Tabel 4.6, maka dapat dibuat grafik perbandingan hasil pengujian algoritma *Stout Code* dan algoritma *Fibonacci Code* untuk *string* heterogen berdasarkan rasio kompresi, waktu kompresi, dan waktu dekompresi. Gambar 4.10, Gambar 4.11 dan Gambar 4.12 menampilkan ketiga grafik tersebut.



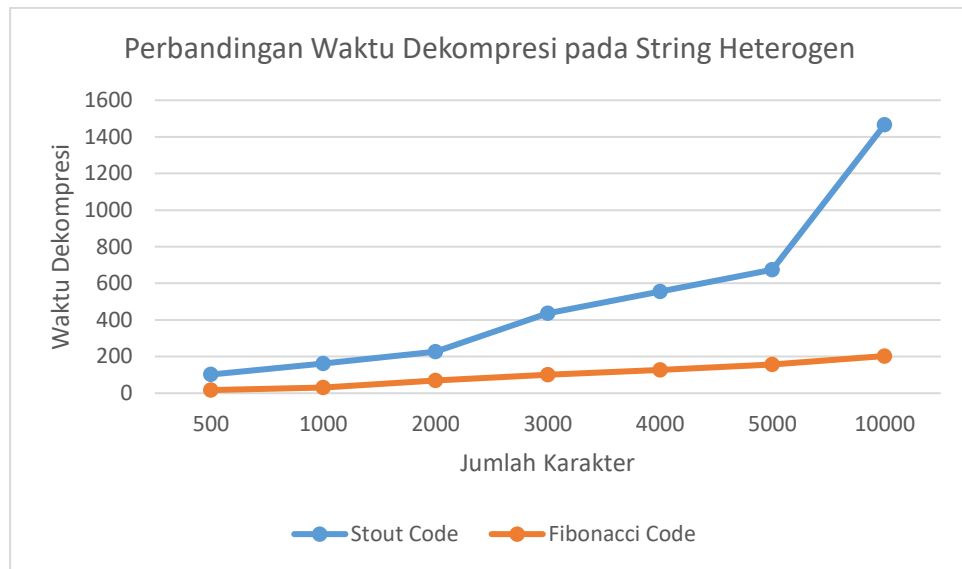
Gambar 4.10. Diagram Perbandingan Rasio Kompresi pada *String* Heterogen

Dari Gambar 4.10, dapat dilihat bahwa algoritma *Stout Code* memiliki nilai rasio kompresi yang lebih besar dari *Fibonacci Code*. Artinya ukuran *file* hasil kompresi menggunakan algoritma *Stout Code* secara umum lebih kecil daripada menggunakan algoritma *Fibonacci Code*. Karena perbandingan keduanya cukup signifikan maka dapat disimpulkan bahwa algoritma *Stout Code* dapat mengompresi *file String* Heterogen menjadi ukuran yang lebih kecil daripada algoritma *Fibonacci Code*.



Gambar 4.11. Diagram Perbandingan Waktu Kompresi pada *String* Heterogen

Dari Gambar 4.11. dapat dilihat bahwa algoritma *Fibonacci Code* memerlukan waktu yang sedikit lebih cepat untuk melakukan proses kompresi daripada algoritma *Stout Code*. Akan tetapi perbedaan waktu kompresi tidak begitu besar. Oleh karena itu, dapat disimpulkan bahwa waktu yang diperlukan algoritma *Stout Code* dan *Fibonacci Code* untuk mengkompresi *file String* Heterogen adalah serupa.



Gambar 4.12. Diagram Perbandingan Waktu Dekompresi pada *String* Heterogen

Dari Gambar 4.12. dapat dilihat bahwa algoritma *Fibonacci Code* memerlukan waktu yang lebih cepat untuk melakukan proses dekompresi daripada algoritma *Stout Code*. Dengan perbedaan waktu yang cukup signifikan, dapat disimpulkan bahwa algoritma *Fibonacci Code* lebih cepat daripada algoritma *Stout Code* dalam melakukan dekompresi *file String* Heterogen.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Setelah pemaparan hasil penelitian, terdapat beberapa kesimpulan yang bisa diambil mengenai perbandingan antara algoritma *Stout Code* dan algoritma *Fibonacci Code* dalam melakukan proses kompresi berkas teks pada aplikasi *Android*. Di antaranya sebagai berikut:

1. Dari hasil pengujian terhadap *string* homogen dan heterogen, ditemukan bahwa algoritma *Stout Code* memberikan rasio kompresi yang lebih baik. Algoritma ini memiliki rata-rata rasio kompresi 1,99 untuk *string* homogen, dan 1,2 untuk *string* heterogen.
2. Dari pengujian terhadap *string* homogen dan heterogen, diketahui bahwa algoritma *Fibonacci Code* lebih unggul dalam hal waktu kompresi. Rata-rata waktu kompresi algoritma ini adalah 13 milidetik untuk *string* homogen, dan 24,14 milidetik untuk *string* heterogen.
3. Hasil pengujian terhadap *string* homogen dan heterogen menunjukkan bahwa algoritma *Fibonacci Code* lebih unggul dalam waktu dekompresi. Rata-rata waktu dekompresi algoritma ini adalah 40,14 milidetik untuk *string* homogen, dan 100,71 milidetik untuk *string* heterogen.
4. Secara umum, pada pengujian kompresi *file* teks homogen dan heterogen didapatkan bahwa algoritma *Stout Code* lebih unggul dari segi ukuran sementara algoritma *Fibonacci Code* lebih unggul dari segi kecepatan.

5.2 Saran

Sejumlah saran yang mungkin bisa membantu penelitian selanjutnya adalah sebagai berikut:

1. Teks yang digunakan pada penelitian ini adalah yang berekstensi *.txt*, untuk penelitian berikutnya mungkin mampu melakukan kompresi pada *file* teks yang berekstensi lain seperti *.pdf*, *.docx* dan lain-lain.
2. Kompresi yang dilakukan pada penelitian ini hanyalah tertuju kepada *file* teks, untuk penelitian berikutnya mungkin dapat mengkompresi *file* dengan bentuk lain seperti audio, video, dan lain-lain.
3. Sistem pada penelitian ini dibangun dengan menggunakan bahasa pemrograman *Kotlin*. Namun, pada penelitian selanjutnya, disarankan untuk mencoba menggunakan bahasa pemrograman yang berbeda seperti *Dart*, *Swift*, dan lain-lain.
4. Sistem yang dibangun pada penelitian ini hanya dapat berjalan pada sistem operasi *Android*, untuk penelitian berikutnya mungkin sistem boleh dibangun untuk dapat berjalan pada sistem operasi *iOS* ataupun keduanya.

DAFTAR PUSTAKA

- Bhattacharyya, S. (2017). Complexity Analysis of a Lossless Data Compression Algorithm using Fibonacci Sequence. *IJIT*, 3(3), 77–82.
- Bhattacharyya, S (2017). Data Compression using Fibonacci Sequence. In T. Kalaiselvi (Ed.), *Computational Methods, Communication Techniques and Informatics* (pp. 151-154). Shanlax Publications.
- Elveny, M., Syah, R., Jaya, I., & Affandi, I. (2020, June). Implementation of Linear Congruential Generator (LCG) Algorithm, Most Significant Bit (MSB) and Fibonacci Code in Compression and Security Messages Using Images. In *Journal of Physics: Conference Series* (Vol. 1566, No. 1, p. 012015). IOP Publishing.
- Hardi, S. M., Angga, B., Lydia, M. S., Jaya, I., & Tarigan, J. T. (2019). Comparative Analysis Run-Length Encoding Algorithm and Fibonacci Code Algorithm on Image Compression. *Journal of Physics: Conference Series*, 1235. <https://doi.org/10.1088/1742-6596/1235/1/012107>
- Hughes, J. (2023). Comparison of lossy and lossless compression algorithms for time series data in the Internet of Vehicles.
- Jayasankar, U., Thirumal, V., & Ponnuram, D. (2021). A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *Journal of King Saud University - Computer and Information Sciences*, 33(2), 119–140. <https://doi.org/10.1016/j.jksuci.2018.05.006>
- Mahmoudi, R., & Zare, M. (2020). Comparison of Compression Algorithms in text data for Data Mining. *International Journal of Advanced Engineering, Management and Science*, 6(6).
- Nasution, S. D. (2019). Data Compression Using Stout Codes. *IJICS*, 3(1), 28–33.
- Pandey, A. K., Kanchan, S., & Verma, A. K. (2023). Applications of Fibonacci Sequences and Golden Ratio. *Journal of Informatics Electrical and Electronics Engineering (JIEEE)*, 4(1), 1-11.
- Rachmawati, D., Budiman, M. A., & Subada, M. A. (2019). Comparison Study of Fibonacci Code Algorithm and Even-Rodeh Algorithm for Data Compression. *Journal of Physics: Conference Series*, 1321(3). <https://doi.org/10.1088/1742-6596/1321/3/032015>
- Rajasekaran, S., Mohan, M., & Prabhakaran, G. (2022). Digital Preservation File Format Tools. *Knowledge Management in Higher Education Institutions*, 157.

- Sayood, K. (2018). *Introduction to Data Compression* (Fifth Edition). Morgan Kaufmann.
- Sitio, A. S. (2018). Text Message Compression Analysis Using the LZ77 Algorithm. *INFOKUM*, 7(1), 16–21.
- Tanjung, A. S., & Nasution, S. D. (2020). Comparison Analysis with Huffman Algorithm and Goldbach Codes Algorithm in File Compression Text Using the Method Exponential Comparison. *The IJICS (International Journal of Informatics and Computer Science)*, 4(1), 29-34.
- Venkata, S. K., Young, P., & Green, A. (2020). *Using machine learning for text file format identification* (No. 4698). EasyChair.

LAMPIRAN

Listing Program

Source code aplikasi dapat dilihat pada link berikut:

<https://github.com/fadhliibrahims/skripsi>

- **StoutCode.kt**

```
package com.example.skripsi
```

```
class StoutCode {
```

```
    fun pow(value: Long, exp: Int): Long {
```

```
        return when {
```

```
            exp > 1 -> value * pow(value,exp-1)
```

```
            exp == 1 -> value
```

```
            else -> 1
```

```
        }
```

```
    }
```

```
fun generateStoutCodeList(amountOfCode: Int, l: Int): ArrayList<String> {
```

```
    var codeList = ArrayList<String>()
```

```
    if (amountOfCode>0 && l>=2) {
```

```
        for(n in 1..amountOfCode) {
```

```
            if (n>=pow(2, l)) {
```

```
                //L = n.toBinaryString().length
```

```

    val L = n.toString(2).length

    //str = n.toBinaryString(L)

    val str = n.toString(2).padStart(L, '0')

    //m = L-1-l

    var m = L-1-l

    while(m>=pow(2, l)) {

        //L = m.toBinaryString().length

        val L = m.toString(2).length

        //str = m.toBinaryString(L)

        val str = m.toString(2).padStart(L, '0')

        //m = L

        m = L

    }

    codeList.add(m.toString(2).padStart(l, '0') + str + '0')

} else {

    //codelist[n] = n.toBinaryString(l)

    codeList.add(n.toString(2).padStart(l, '0') + '0')

}

}

} else {

    print("Selesai")

```

```

    }

    return codeList

}

}

```

- **FibonacciCode.kt**

```
package com.example.skripsi
```

```
class FibonacciCode {
```

```
    private val N = 30
```

```
    private val fib = IntArray(N)
```

```
    private fun largestFiboLessOrEqual(n: Int): Int{
```

```
        fib[0] = 1
```

```
        fib[1] = 2
```

```
        var i = 2
```

```
        while (fib[i-1] <= n) {
```

```
            fib[i] = fib[i-2] + fib[i-1]
```

```
            i = i+1
```

```
        }
```

```
        return (i-2)
```

```
    }
```

```

private fun fibonacciEncoding(n: Int): String{

    val index = largestFiboLessOrEqual(n)

    var codeword = CharArray(index+2)

    var i = index

    var x = n

    while (x > 0) {

        codeword[i] = '1'

        x = x-fib[i]

        i = i-1

        while (i >= 0 && fib[i] > x)

        {

            codeword[i] = '0'

            i = i - 1

        }

    }

    codeword[index + 1] = '1'

    return String(codeword)

}

fun generateFibonacciCodeList(n: Int): ArrayList<String> {

```

```

var codeList = ArrayList<String>()

for(i in 1..n) {

    codeList.add(fibonacciEncoding(i))

}

return codeList

}

}

• Kompres.kt

package com.example.skripsi

class Kompres {

    fun kompresText(text: String, algorithm: Int): String {

        val asciiStringBit = textToAscii(text)

        val charset = getCharsetFromText(text)

        val freqList = getFreqOfEachCharFromText(text, charset)

        val sortedCharset = insertionSort(charset, freqList)

        var compressedBit = ""

        if (algorithm == 0) {

            //Generate Stout Code

            val stoutCode = StoutCode()

```

```

val stoutCodeList = stoutCode.generateStoutCodeList(charset.size, 2)

//Create Encoding Table

val encodingTable = HashMap<Char, String>()

for(i in 0..sortedCharset.size-1) {

    encodingTable[sortedCharset[i]] = stoutCodeList[i]

}

//Compression

for(i in 0..text.length-1) {

    compressedBit = compressedBit + encodingTable[text[i]]

}

} else if (algorithm == 1) {

    //Generate Fibonacci Code

    val fibonacciCode = FibonacciCode()

    val fibonacciCodeList = fibonacciCode.generateFibonacciCodeList(charset.size)

    //Create Encoding Table

    val encodingTable = HashMap<Char, String>()

    for(i in 0..sortedCharset.size-1) {

        encodingTable[sortedCharset[i]] = fibonacciCodeList[i]

    }

```

```

//Compression

for(i in 0..text.length-1) {

    compressedBit = compressedBit + encodingTable[text[i]]

}

}

//Padding Bit and Flag Bit

val padLength = 8-(compressedBit.length%8)

val paddingBit = "0".repeat(padLength)

val flagBit = padLength.toString(2).padStart(8, '0')

compressedBit = compressedBit + paddingBit + flagBit

//Convert compressedBit to Text

var    compressedText    =    asciiToText(compressedBit)    +    "|*|"    +
sortedCharset.joinToString("")

return compressedText

}

private fun textToAscii(text: String): String {

    var asciiStringBit = ""

    for(char in text) {

        asciiStringBit += char.code.toString(2).padStart(8, '0')

```



```

    }

    return asciiStringBit
}

private fun asciiToText(asciiStringBit: String): String {

    var text = ""

    var asciiCode = ""

    for(char in asciiStringBit) {

        asciiCode = asciiCode + char

        if(asciiCode.length == 8) {

            text = text + Integer.parseInt(asciiCode, 2).toChar().toString()

            asciiCode = ""

        }

    }

    return text
}

```

```

private fun getCharsetFromText(text: String): ArrayList<Char> {

    var charset = ArrayList<Char>()

    for(char in text) {

        if(!charset.contains(char)) {

            charset.add(char)

        }

    }

}

```

```

    }

}

return charset

}

```

```

private fun getFreqOfEachCharFromText(text: String, charset: ArrayList<Char>):
ArrayList<Int> {

    var freqList = ArrayList<Int>()

    for(char in charset) {

        val charFreq = text.split(char).size - 1

        freqList.add(charFreq)

    }

    return freqList

}

```

```

private fun insertionSort(charset: ArrayList<Char>, freqList:ArrayList<Int>):
ArrayList<Char>{

    if (freqList.isEmpty() || freqList.size<2){

        return charset

    }

    for (count in 1..freqList.count() - 1){

        val freq = freqList[count]

        val char = charset[count]

```

```

var i = count

while (i>0 && freq > freqList[i - 1]){

    freqList[i] = freqList[i - 1]

    charset[i] = charset[i - 1]

    i -= 1

}

freqList[i] = freq

charset[i] = char

}

return charset

}

}

```

- **Dekompres.kt**

```
package com.example.skripsi
```

```

class Dekompres {

    fun dekompresText(compressedText: String, algorithm: Int): String {

        //Get Charset from Compressed Text

        var decompressedText = ""

        val parts = compressedText.split("|*|")

        val compressedText2 = parts[0]

```

```

val charset2 = parts[1]

//Convert compressed text to compressed bit
var compressedBit2 = textToAscii(compressedText2)

//Remove padding bit and flag bit

val      flagBit2      =      compressedBit2.substring(compressedBit2.length-8,
compressedBit2.length)

val padLength2 = Integer.parseInt(flagBit2, 2)

compressedBit2 = compressedBit2.dropLast(padLength2+8)

if (algorithm == 0) {
    //Generate Table Encoding

    val l2 = 2

    val amountOfCode2 = charset2.length

    val stoutCode = StoutCode()

    val stoutCodeList2 = stoutCode.generateStoutCodeList(amountOfCode2, l2)

    val encodingTable2 = HashMap<String, Char>()

    for(i in 0..charset2.length-1) {

        encodingTable2[stoutCodeList2[i]] = charset2[i]

    }

    println(encodingTable2)

```

```

//Decompression

var counter = 0

var x = ""

var y = ""

var bigL = 0

while(counter < compressedBit2.length) {

    x = compressedBit2.substring(counter, counter+l2+1)

    y = compressedBit2.substring(counter, counter+l2)

    counter = counter + l2+1

    while(encodingTable2.containsKey(x)==false) {

        bigL = binaryToDecimal(y.toLong()) + 1 + l2

        x = x + compressedBit2.substring(counter, counter+bigL)

        y = compressedBit2.substring(counter-1, counter+bigL-1)

        counter = counter + bigL

    }

    decompressedText = decompressedText + encodingTable2[x]

}

} else if (algorithm == 1) {

    //Generate Table Encoding

    val amountOfCode2 = charset2.length

```

```

val fibonacciCode = FibonacciCode()

val fibonacciCodeList2 =
    fibonacciCode.generateFibonacciCodeList(amountOfCode2)

val encodingTable2 = HashMap<String, Char>()

for(i in 0..charset2.length-1) {

    encodingTable2[fibonacciCodeList2[i]] = charset2[i]

}

//Decompression

var charCounter = 2

var stringBit = compressedBit2.substring(0, charCounter)

while(charCounter <= compressedBit2.length) {

    if(stringBit.substring(stringBit.length-2, stringBit.length) == "11") {

        decompressedText = decompressedText +
            encodingTable2.getValue(stringBit).toString()

        if(charCounter < compressedBit2.length) {

            stringBit = compressedBit2.substring(charCounter, charCounter+2)

        }

        charCounter = charCounter + 2

    } else {

        stringBit = stringBit + compressedBit2.substring(charCounter,
            charCounter+1)
    }
}

```

```

        charCounter = charCounter + 1
    }
}
}

return decompressedText
}

```

```

private fun textToAscii(text: String): String {
    var asciiStringBit = ""

    for(char in text) {
        asciiStringBit += char.code.toString(2).padStart(8, '0')
    }

    return asciiStringBit
}

```

```

private fun asciiToText(asciiStringBit: String): String {
    var text = ""

    var asciiCode = ""

    for(char in asciiStringBit) {
        asciiCode = asciiCode + char

        if(asciiCode.length == 8) {
            text = text + Integer.parseInt(asciiCode, 2).toChar().toString()
        }
    }

    return text
}

```

```

        asciiCode = ""
    }
}

return text
}

```

```

private fun getCharsetFromText(text: String): ArrayList<Char> {

    var charset = ArrayList<Char>()

    for(char in text) {

        if(!charset.contains(char)) {

            charset.add(char)

        }

    }

    return charset
}

```

```

private fun getFreqOfEachCharFromText(text: String, charset: ArrayList<Char>):
ArrayList<Int> {

    var freqList = ArrayList<Int>()

    for(char in charset) {

        val charFreq = text.split(char).size - 1

        freqList.add(charFreq)

    }
}

```



```

    return freqList
}

```

```

private fun binaryToDecimal(num: Long): Int {

    var num = num

    var decimal = 0

    var i = 0

    var remainder: Long

    while (num.toInt() != 0) {

        remainder = num % 10

        num /= 10

        decimal += (remainder * Math.pow(2.0, i.toDouble())).toInt()

        ++i

    }

    return decimal

}

```

```

private fun insertionSort(charset: ArrayList<Char>, freqList: ArrayList<Int>):
ArrayList<Char>{

    if (freqList.isEmpty() || freqList.size<2){

        return charset

    }
}

```

```
}  
  
for (count in 1..freqList.count() - 1){  
  
    // println(items)  
  
    val freq = freqList[count]  
  
    val char = charset[count]  
  
    var i = count  
  
    while (i>0 && freq > freqList[i - 1]){  
  
        freqList[i] = freqList[i - 1]  
  
        charset[i] = charset[i - 1]  
  
        i -= 1  
  
    }  
  
    freqList[i] = freq  
  
    charset[i] = char  
  
}  
  
return charset  
  
}  
  
}
```

CURRICULUM VITAE

Nama : Fadhli Ibrahim Siregar
Tempat/Tanggal Lahir : Kisaran, 12 Oktober 1999
Jenis Kelamin : Laki-laki
Agama : Islam
Kewarganegaraan : Indonesia
Alamat : Bunut Barat Lk. II
No. HP : 085156025167
Email : formalfadhli@outlook.com

RIWAYAT PENDIDIKAN

2017-2023 : Program Studi S-1 Ilmu Komputer, Fakultas Ilmu
Komputer dan Teknologi Informasi, Universitas
Sumatera Utara, Medan
2014-2017 : MA Negeri Kisaran
2011-2014 : MTs Negeri Kisaran
2005-2011 : SD Alwashliyah 80 Kisaran