Jurnal Kajian Ilmiah Teknologi Informasi dan Komputer
ISSN 2962-9055 (Media Online)
Vol 3, No 1, November 2024
Hal 24-33
https://journal.grahamitra.id/index.php/jutik

Implementasi Algoritma Additive Code Untuk Mengkompresi File PDF Pada Aplikasi E-Archive

Fitri Ayu Ningsih

Program Studi Teknik Informatika, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Budi Darma,
Jalan Sisingamanganraja No. 338, Medan, Sumatera Utara, Indonesia
Email: fitriayuningsih456@gmail.com

Abstrak

Permasalahan kebutuhan ruang penyimpanan pada aplikasi *E-Archive* merupakan salah satu masalah utama yang dapat mempengaruhi kinerja dari aplikasi *E-Archive* sendiri. Hal ini disebabkan karena banyaknya *file* PDF berukuran besar yang tersimpan di dalam *database* aplikasi *E-Archive*. Salah satu solusi yang dapat dilakukan untuk mengatasi permasalahan tersebut adalah melakukan kompresi pada *file* PDF. Proses kompresi pada penelitian ini dilakukan berdasarkan algoritma *Additive Code*. Berdasarkan hasil pengujian yang dilakukan diperoleh *Ratio of Compression* (RC) sebesar 1,33, *Compression ratio* (Cr) berkurang sebanyak 75%, *Redundancy* (Rd) berkurang sebanyak 25% dan *Space saving* (Ss) berkurang sebanyak 25%. Hasil menunjukkan bahwa kebutuhan ruang penyimpanan *database* aplikasi *E-Archive* lebih optimal daripada sebelum diterapkan proses kompresi.

Kata Kunci: Kompresi; File PDF; Aplikasi E-Archive; Database; Algoritma Additive Code

Abstract

Problems with the need for storage space in the E-Archive application is one of the main problems that can affect the performance of own E-Archive application. This is due to the large number of PDF files large files stored in the E-Archive application database. One of solutions that can be done to overcome this problem are compress PDF files. The compression process in this study was carried out based on an algorithm Additive Code. Based on the results of the tests performed, the Ratio of Compression (RC) of 1.33, Compression ratio (Cr) reduced by as much 75%, Redundancy (Rd) is reduced by 25% and Space saving (Ss) is reduced as much as 25%. The results show that the need for storage space the E-Archive application database is more optimal than before the process was implemented compression.

Keywords: Compression; PDF Files; E-Archive Application; Database; Additive Code Algorithm

1. PENDAHULUAN

Aktifitas surat menyurat dan dokumen merupakan aktifitas yang selalu ada dalam sektor perkantoran baik instansi pemerintahan, perusahaan dan instansi pendidikan. Proses surat menyurat bertujuan menyampaikan informasi terkait administratif dan pelaksanaan kegiatan. Namun di era serba digital, banyak instansi masih memanfaatkan cara manual yang dianggap kurang efektif dalam kegiatan surat menyurat sehingga surat atau dokumen sering mengalami kerusakan. Untuk mewujudkan tata kelola persuratan yang baik, efektif dan efisien, salah satunya adalah dibangun aplikasi *E-Archive*

E-Archive atau *electronic archive* adalah suatu sistem yang memberikan informasi berupa sebuah dokumen yang direkam dan disimpan menggunakan teknologi komputer berbentuk dokumen elektronik dengan tujuan membantu dalam penyimpanan dokumen secara digital sehingga data menjadi terkomputerisasi dan tidak mudah rusak, memudahkan dalam pencarian data dokumen sehingga proses pencarian dokumen dapat dilakukan dengan lebih mudah dan cepat. Aplikasi *E-Archive* dirancang sebagai salah satu media alternative dari pemanfaatan teknologi yang digunakan untuk mempelajari suatu ilmu, materi, pengetahuan dan wawasan yang bersumber. Aplikasi *E-Archive* menampilkan banyak jenis-jenis arsip yang diupload, maka dibutuhkan ruang penyimpanan yang cukup luas untuk menampung data arsip tersebut. *File* PDF yang berukuran besar membutuhkan waktu lama dalam proses pengiriman data dan menghabiskan ruang penyimpanan. Peran media penyimpanan data dalam aplikasi *E-Archive* sangat penting, karena yang mengatur jalannya suatu proses dan menyimpan data dengan aman. Berdasarkan penelitian terdahulu terkait dengan media penyimpanan menyimpulkan bahwa dengan adanya media penyimpanan data yang terintegrasi ke dalam sebuah aplikasi dapat meminimalisir hilangnya data-data yang ada serta dapat membantu mempercepat kinerja pada aplikasi tersebut dikarenakan memudahkan sekolah untuk mendapatkan informasi yang dibutuhkan secara tepat, cepat dan akurat[1]. *Database* mempunyai peran penting dalam perangkat guna mengumpulkan fakta data arsip secara terintegrasi. Peranan *database* pada aplikasi *E-Archive* yaitu mengelompokkan sebuah data arsip, memudahkan identifikasi data, menghindari duplikasi data dan inkosistensi data, memudahkan akses penyimpanan data dan menyediakan data yang relevan bertujuan menjaga kualitas data, memecahkan masalah penyimpanan data yang terbatas serta mendukun aplikasi aplikasi *E-Archive* yang membutuhkan ruang penyimpanan.

Sebagai upaya untuk menghemat ruang penyimpanan yaitu dengan dilakukan proses kompresi pada data di dalam *database*. Berdasarkan penelitian terdahulu terkait pemanfaat teknik kompresi menyimpulkan bahwa teknik kompresi yang ada sekarang memungkinkan data dikompresi sehingga ukurannya menjadi jauh lebih kecil dari pada ukuran asli, mengurangi waktu transmisi data dikirim, dan tidak banyak menghabiskan ruang media penyimpanan dan hasil dari aplikasi kompresi data teks tersebut menampilkan data teks yang sudah diolah pada aplikasi dengan proses kompresi maupun dekompresi dan kemudian disimpan di *database*[2].Salah satu algoritma kompresi yang dapat membantu berjalannya proses kompresi adalah algoritma *additive code*. Algoritma *additive code* dapat membantu dalam mengkompresi *file* PDF karena algoritma ini bersifat *lossless*. Algoritma *additive code* adalah algoritma yang mencakup beberapa algoritma kompresi yaitu algoritma *golbach code* dan algoritma *elias gamma code* yang dimana memungkinkan

Jurnal Kajian Ilmiah Teknologi Informasi dan Komputer
ISSN 2962-9055 (Media Online)
Vol 3, No 1, November 2024
Hal 24-33
https://journal.grahamitra.id/index.php/jutik

lebih efisien dalam mengkompresi sebuah *file* pdf. Berdasarkan penelitian terdahulu terkait terkait cara kerja algoritma *additive code* menyimpulkan bahwa setelah diimplementasikan untuk mengkompres *file* PDF hasilnya adalah rasio kompresi menjadi lebih rendah[3].

Permasalahan utama yang sering timbul pada aplikasi *E-Archive* adalah kebutuhan ruang penyimpanan *file database* yang cukup besar. Namun dengan diterapkannya proses kompresi terhadap *file* PDF yang disimpan di dalam *database*,maka sangat bermanfaat dan membantu untuk mengakses aplikasi *E-Archive* tanpa harus memikirkan ketersediaan ruang penyimpanan yang terbatas.Proses kompresi pada penelitian ini dilakukan pada saat *file* PDF diuplod ke dalam aplikasi *E-Archive* lalu disimpan di dalam *database* aplikasi tersebut. Jika pengguna mengakses data arsip, maka otomatis sistem akan melakukan proses dekompresi untuk menampilkan data aslinya.

2. METODOLOGI PENELITIAN

2.1 Kompresi

Kompresi adalah proses memampatkan atau memperkecil data yang berukuran besar. Kompresi data atau *data compression* merupakan sebuah teknik dalam ilmu computer untuk mengompresi data sehingga membutuhkan lebih sedikit ruang penyimpanan, membuat proses penyimpanan lebih efisien dan mengurangi waktu pertukaran data[3].

Proses kompresi dapat dilakukan terhadap sebuah teks/biner, dokumen (docx, pdf, xlss, pptx), gambar (JPEG, PNG, TIFF), audio (MP3, ACC, RMA, WMA) dan video (MPEG, H261, H263). Kompresi data terbagi 2 berdasarkan jenisnya[3][4], yaitu:

1. *Lossy Compression* adalah teknik kompresi dimana terdapat data yang hilang pada saat melakukan proses kompresi, dimana hasil kualitas data jauh lebih rendah dari kualitas data aslinya. Jenis kompresi *lossy* digunakan untuk data organic seperti sinyal audio dan gambar yang tidak mungkin dalam kasus data teks. Rasio kompresi pada *lossy* bias lebih tinggi.
2. *Lossless Compression* adalah teknik kompresi dimana tidak menghilangkan data selama proses kompresi sehingga kualitas data yang dihasilkan tidak berkurang dan hasil dari data dekompresi yang terkompresi sama seperti data aslinya. Kompresi *lossless* dapat diterapkan ke format *file* apa saja yang dapat meningkatkan kinerja rasio. Rasio kompresi pada *lossless* cenderung lebih rendah dibandingkan dengan kompresi *lossy*.

2.2 Algoritma Additive Code

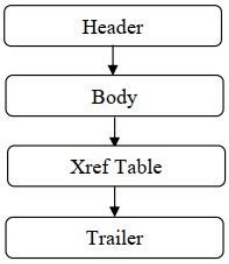
Algoritma *additive code* merupakan algoritma yang mencakup beberapa algoritma kompresi yaitu *golbach code* dan *elias gamma code* yang dimana memungkinkan lebih efisien dalam mengkompresin sebuah *file*. Algoritma *additive code* adalah salah satu teknik kompresi yang dapat memperkecil suatu data berdasarkan dengan karakter pada objek yang akan dilakukan proses kompresi dan mengarah pada kode yang sederhana dan efisien menunjukkan bahwa barisan lain dapat digunakan dengan cara yang sama. Algoritma *additive code* memiliki *codeword* yang dimana dapat membantu proses kompresi *file* dengan perhitungan secara manual. Berikut merupakan langkah-langkah dalam membangun *codeword* algoritma *additive code*[5]:

1. Tentukan nilai n merupakan nilai yang diturunkan dari *sequence* algoritma *golbach code.*
2. Tentukan nilai sum yang merupakan penjumlahan 2 *sequence* algoritma *golbach code* yang menghasilkan nilai n(a^i+ a^j=n)
3. *Indexes* yaitu index (posisi urutan) dari a^i, a^j
4. *Pair* yaitu selisih index dari a^i dan a^j adalah (a^i(a^j- a^i +1)
5. *Codeword* adalah hasil algoritma *elias gamma code* dari a^i dan a^j dengan ketentuan:
   a. Menentukan bilangan bulat N terbesar sehingga $2^N \leq n < 2^{N+1}$ ditulis $n = 2^N + L$
   b. Merubah nilai n menjadi bilangan biner, lalu hilangkan 1 bit paling kiri
   c. Kodekan N dalam bentuk *unary* sebagai N nol diikuti oleh 1 atau N 1 diikuti oleh 0
   d. Menambahkan sisa digit biner n dibelakang kode *unary* yang telah dihasilkan.
6. *Length* merupakan hasil jumlah digit biner yang ada pada *codeword*.

2.3 Portable Document Format

PDF merupakan singkatan dari *Portable Document Format* dan *file* dengan ekstensi ini dikembangkan oleh *adobe system. File* PDF memiliki ukuran yang besar, sehingga membutuhkan waktu lama dalam proses pengiriman dan menghabiskan banyak ruang penyimpanan. Format PDF digunakan untuk merepresentasikan dokumen 2D diantaranya adalah teks, huruf, gambar dan grafik. Format PDF adalah format dokumen yang sangat popular karena tidak bergantung pada perangkat lunak. PDF memiliki lebih banyak fungsi bukan hanya teks melainkan dapat menggunakan objek lain seperti gambar dan elemen multimedia lainnya bahkan PDF dapat dibuat kata sandi. Struktur dasar *file* PDF disajikan pada Gambar 1.

Jurnal Kajian Ilmiah Teknologi Informasi dan Komputer
ISSN 2962-9055 (Media Online)
Vol 3, No 1, November 2024
Hal 24-33
https://journal.grahamitra.id/index.php/jutik



Gambar 1. Struktur *file* PDF

3. HASIL DAN PEMBAHASAN

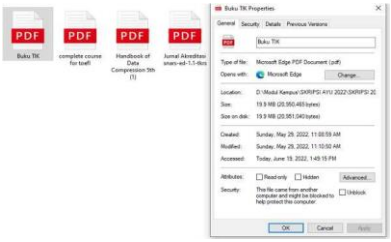
Hasil dan pembahasan pada penelitian ini mengacu pada tahapan proses kompresi dan dekompresi terhadap *file* PDF yang berukuran besar dengan penerapan dari algoritma *additive code*.

3.1 Analisa

Analisa dalam penelitian ini merupakan perhitungan dan perancangan perangkat lunak *E-Archive* yang di dalam aplikasi tersebut terdapat proses kompresi dan dekompresi terhadap *file* PDF dan penulis menggunakan algoritma *additive code*. Penelitian ini dilakukan untuk proses kompresi *file* PDF pada aplikasi *E-Archive* menerapkan algoritma *additive code*, *file* PDF yang pada umumnya memiliki ukuran yang relatif besar. Oleh karena itu, hasil penelitian ini diharapkan dapat membantu memperkecil ukuran *file* PDF. Proses awal yang dilakukan untuk mengimplementasikan metode ini adalah siapkan *file* PDF yang akan dikompresi kemudian mengkonversi *file* PDF ke nilai *hexadecimal* menggunakan aplikasi *binary viewer*, setelah didapat nilai *hexadecimal file* PDF maka akan dilakukan proses kompresi. Hasil kompresi tersebut kembali menjadi *file* PDF, kemudian disimpan ke dalam *database.* Dekompresi dilakukan setelah *file* PDF dikompresi dan tentu saja perlu dipulihkan lagi ke bentuk semula.

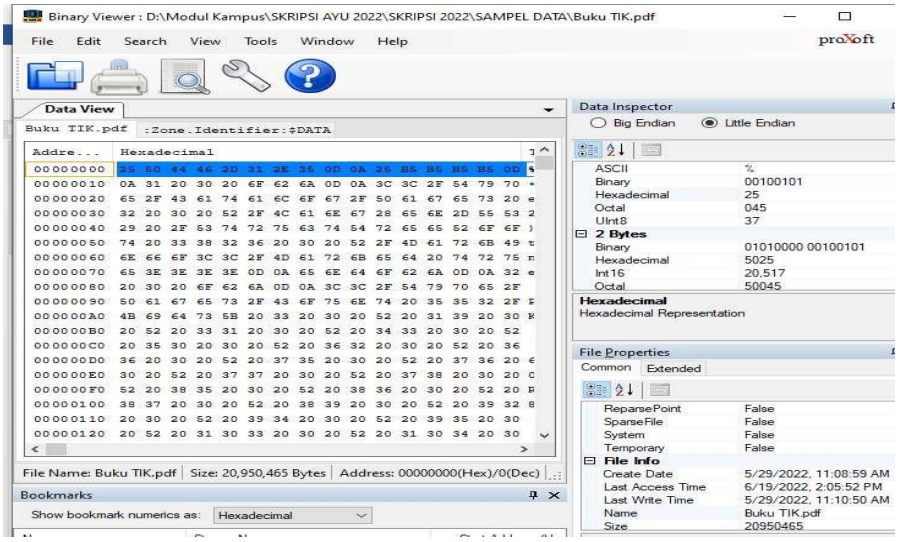
3.1.1 Penerapan Algoritma Additive Code

Sebelum *file* PDF dilakukan proses kompresi, penulis terlebih dahulu siapkan *file* PDF yang akan dikompresi dengan menerapkan algoritma *additive code*.



Gambar 2. Sampel *File* PDF

Sebelum *file* dikompresi terlebih dahulu dilakukan pembacaan biner yang terdapat pada *file* PDF untuk mendapatkan data berupa nilai *hexadecimal*. Hasil nilai *hexadecimal file* PDF ditampilkan pada Gambar 3.



Gambar 3. Nilai *Hexadecimal* Sampel *File* PDF

Jurnal Kajian Ilmiah Teknologi Informasi dan Komputer
ISSN 2962-9055 (Media Online)
Vol 3, No 1, November 2024
Hal 24-33
https://journal.grahamitra.id/index.php/jutik

Berdasarkan gambar nilai *hexadecimal* di atas, penulis mengambil 16 nilai *hexadecimal* untuk dilakukan proses kompresi *file* PDF.

Tabel 1. Nilai Bilangan *Hexadecimal*

25	50	44	46	2D	31	2E	35
0D	0A	25	B5	B5	B5	B5	0D

Tabel 1 menunjukkan bahwa hasil bilangan *hexadecimal file* PDF yang dihitung secara manual akan diurutkan berdasarkan frekuensi kemunculan.

Tabel 2. Data Sebelum Dikompresi Menggunakan Algoritma *Additive Code*

n	Hexadecimal	Biner	Bit	Frekuensi	Bit x Frekuensi
1	B5	1011 0101	8	4	32
2	25	0010 0101	8	2	16
3	0D	0000 1101	8	2	16
4	50	0101 0000	8	1	8
5	44	0100 0100	8	1	8
6	46	0100 0110	8	1	8
7	2D	0010 1101	8	1	8
8	31	0011 0001	8	1	8
9	2E	0010 1110	8	1	8
10	35	0011 0101	8	1	8
11	0A	0000 1010	8	1	8
		Total			128 bit

a. Proses Kompresi

Setelah didapatkan jumlah bit dari perhitungan nilai *hexadecimal* dan frekuensi kemunculan, selanjutnya menghitung jumlah bit dari perhitungan nilai *hexadecimal* dan nilai biner yang dihimpun ke dalam table kebenaran dengan menerapkan algoritma *additive code.*

Tabel 3. Data Setelah Dikompresi Menggunakan Algoritma *Additive Code*

N	Hexadecimal	Frekuensi	Codeword	Bit	Bit x Frekuensi
1	B5	4	1010	4	16
2	25	2	0101	4	8
3	0D	2	1011	4	8
4	50	1	010010	6	6
5	44	1	100100	6	6
6	46	1	010011	6	6
7	2D	1	100101	6	6
8	31	1	01000100	8	8
9	2E	1	100110	6	6
10	35	1	011011	6	6
11	0A	1	100111	6	6
		Total			82 bit

Setelah kode berhasil di*encode* berdasarkan perhitungan algoritma *additive code*, tahap selanjutnya adalah menyusun kembali *string bit* yang telah dihasilkan dari proses kompresi sesuai dengan posisi karakter pada nilai *hexadecimal*.

Tabel 4. *String Bit* Hasil Kompresi Menggunakan Algoritma *Additive Code*

25	**50**	**44**	**46**
0101	010010	100100	010011
2D	**31**	**2E**	**35**
100101	01000100	100110	011011
0D	**0A**	**25**	**B5**
1011	100111	0101	1010
B5	**B5**	**B5**	**0D**
1010	1010	1010	1011

Berdasarkan Tabel 4 *string bit* yang dihasilkan dari proses kompresi *file* PDF menggunakan algoritma *additive code* menghasilkan 82 bit terkompresi yaitu "01010100101001000100111

Jurnal Kajian Ilmiah Teknologi Informasi dan Komputer
ISSN 2962-9055 (Media Online)
Vol 3, No 1, November 2024
Hal 24-33
https://journal.grahamitra.id/index.php/jutik

00101010001001001100110111011100111010110101010101010101011". Kemudian sebelum didapatkan hasil keseluruhan akhir proses kompresi dilakukan penambahan *string bit* yaitu dengan cara *padding* dan *flagging*. Jika sisa bagi panjang *string bit* habis dibagi 8 maka tambahkan 1 yang dinyatakan sebagai bit akhir menjadi 00000001. Sedangkan jika sisa bagi panjang *string bit* tidak habis dibagi 8 atau memiliki sisa hasil bagi n (1,2,3,4,5,6,7) maka tambahkan 0 sebanya 7 – n + "1" di akhir *string bit*. Lalu untuk *flagging* tambahkan bilangan biner dari 9 – n. Dikarenakan jumlah *string bit* 82 dan tidak habis dibagi 8 atau memiliki sisa bagi sebanyak 2 (82 mod 8 = 10 sisa 2) maka harus dilakukan penambahan *string bit* yaitu *padding* dan *flagging* seperti pada Tabel 5.

Tabel 5. Penambahan *Padding* dan *Fagging*

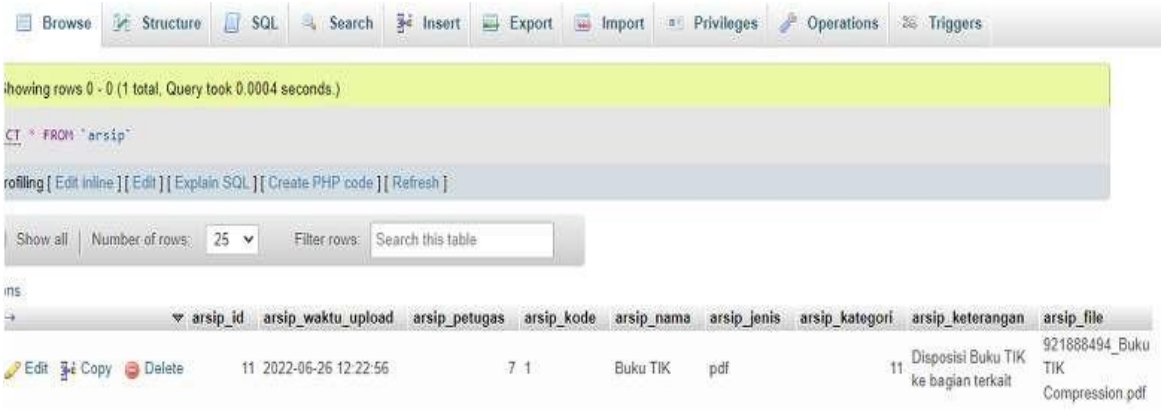
Padding	***Flagging***
82 mod 8 = 10 sisa 2 7 – n + "1" = 7 – 2 + "1" = **000001** (karena 7 – 2 = 5, maka bit "0" ada 5 yang ditambahkan + bit "1").	9 – n 9 – 2 = 7 biner dari 7 = **00000111**

Maka total keseluruhan *string bit* setelah dilakukan penambah bit dengan cara *padding* dan *flagging* adalah 82+14=96 bit "01010100101001000100111 00101010001001001100110111011100111010110101010101010101011**00000100000111**". Sehingga dari keseluruhan bit tersebut dapat dibagi menjadi per 8 bit dan dihimpun ke dalam tabel pada Tabel 6.

Tabel 6. Pengelompokan Bit

01010100	10100100	01001110	01010100
01001001	10011011	10111001	11010110
10101010	10101010	11**000001**	**00000111**

Langkah akhir dari proses kompresi adalah nilai biner yang telah didapat pada Tabel 6 akan diubah menjadi suatu karakter dengan terlebih dahulu mengubahnya menjadi nilai desimal dengan menggunakan kode ASCII maka didapat *string* karakter terkompresi "**T¤NTI ¹Ö^{aa}Á**". *String* karakter terkompresi yang dihasilkan disimpan ke dalam *database* aplikasi *E-Archive* yang dikemas dalam bentuk *file* dengan ekstensi PDF.



Gambar 4. *File* PDF Tersimpan Didalam *Database*

Selanjutnya mengukur hasil kompresi data berdasarkan dengan parameter yang telah ditentukan. Semakin kecil hasil kompresinya maka semakin baik kualitas kinerja kompresinya.

1. *Ratio of Compression* (RC)
   $RC = \frac{Ukuran\ Data\ Sebelum\ Dikompresi}{Ukuran\ Data\ Setelah\ Dikompresi}$
   $RC = \frac{128}{96} = 1{,}33$
2. *Compression Ratio* (CR)
   $CR = \frac{Ukuran\ Data\ Setelah\ Dikompresi}{Ukuran\ Data\ Sebelum\ Dikompresi}\ x\ 100\%$
   $CR = \frac{96}{128}\ x\ 100\% = 75\%$
3. *Redundancy* (Rd)
   $Rd = \frac{File\ Sebelum\ Dikompresi - File\ Setelah\ Dikompresi}{Ukuran\ File\ Sebelum\ Dikompresi}\ x\ 100\%$
   $Rd = \frac{128-96}{128}\ x\ 100\% = 25\%$
4. *Space Saving* (Ss)
   $Ss = \left((1 - \frac{File\ Setelah\ Dikompresi}{File\ Sebelum\ Dikompresi})\right)\ x\ 100\%$

Jurnal Kajian Ilmiah Teknologi Informasi dan Komputer
ISSN 2962-9055 (Media Online)
Vol 3, No 1, November 2024
Hal 24-33
https://journal.grahamitra.id/index.php/jutik

$$Ss= \left((1 - \frac{96}{128})\right) \; x \; 100\% = (1 - 0.75)x \; 100\% = 25\%$$

Berdasarkan perhitungan di atas dapat disimpulkan bahwa persentase hasil *Compression Ratio* (CR) dengan menggunakan algoritma *additive code* sebesar 75%

b. Proses Dekompresi dilakukan pada saat petugas/*user* mengunduh *file* arsip darin aplikasi *E-Archive*. Saat proses unduh dilakukan maka aplikasi *E-Archive* melakukan pembacaan nilai *hexadecimal file* yang diunduh kemudia dilakukan proses dekompresi. Dikarenakan isi dari *file* PDF berupa karakter *string* terkompresi "**T¤NTI ¹Öªª Á**" maka lakukan proses kompresi awal dengan mengubah karakter string tersebut dengan nilai biner. Sebelum diubah ke nilai biner, ubah terlebih dahulu karakter *string* menjadi nilai decimal dan didapat nilai biner "01010100101001000100111 00101010001001001100110111011100111010110101010101010101011**00000100000111**". Setelah didapat nilai biner, kembalikan nilai biner menjadi *string bit* semula dengan menghilangkan biner *padding* dan *flagging*. Lakukan pembacaan pada 8 bit terakhir, hasil pembacaan dengan n, hilangkan bit pada bagian akhir sebanyak 7+n, diketahui n = 1 maka 7+1=8. 8 bit terakhir = **00000111** = 7 = n. 7+n= 7+7=14. Hilangkan bit sebanyak 14 bit sehingga menjadi "01010100101001000100111 00101010001001001100110111011100111010110101010101010101011" Setelah mengembalikan jumlah bit semula menjadi 82 bit, selanjutnya melakukan pembacaan *string bit* dengan algoritma *additive code* menggunakan teknik *Brute Force*.

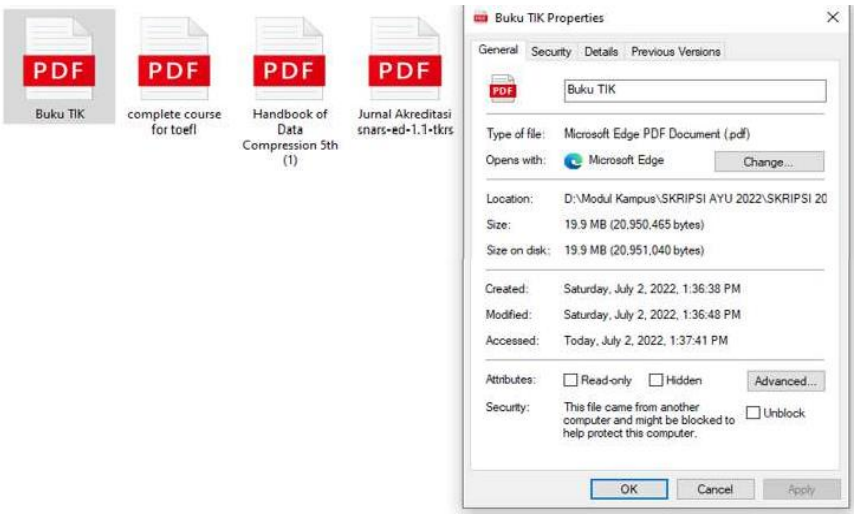
Tabel 7. Pengecekan Bit Dekompresi

Indeks	Nilai Biner	Keterangan	Karakter
1	0	Tidak Ada	
2	01	Tidak Ada	
3	010	Tidak Ada	
4	0101	Ada	25
5	1	Tidak Ada	
6	10	Tidak Ada	
7	101	Tidak Ada	
8	1011	Ada	0D
9	0	Tidak Ada	
10	01	Tidak Ada	
11	010	Tidak Ada	
12	0100	Tidak Ada	
13	01001	Tidak Ada	
14	010010	Ada	50
15	1	Tidak Ada	
16	10	Tidak Ada	
17	100	Tidak Ada	
18	1001	Tidak Ada	
19	10010	Tidak Ada	
20	100100	Ada	44
21	0	Tidak Ada	
22	01	Tidak Ada	
23	010	Tidak Ada	
24	0100	Tidak Ada	
25	01001	Tidak Ada	
26	010011	Ada	46
27	1	Tidak Ada	
28	10	Tidak Ada	
29	101	Tidak Ada	
30	1010	Ada	B5
31	1	Tidak Ada	
32	10	Tidak Ada	
33	100	Tidak Ada	
34	1001	Tidak Ada	
35	10010	Tidak Ada	
36	100101	Ada	2D
37	1	Tidak Ada	
38	10	Tidak Ada	
39	101	Tidak Ada	
40	1010	Ada	B5
41	0	Tidak Ada	
42	01	Tidak Ada	

ISSN 2962-9055 (Media Online)
Vol 3, No 1, November 2024
Hal 24-33
https://journal.grahamitra.id/index.php/jutik

43	010	Tidak Ada	
44	0100	Tidak Ada	
45	01000	Tidak Ada	
46	010001	Tidak Ada	
47	0100010	Tidak Ada	
48	01000100	Ada	31
49	0	Tidak Ada	
50	01	Tidak Ada	
51	010	Tidak Ada	
52	0101	Ada	25
53	1	Tidak Ada	
54	10	Tidak Ada	
55	101	Tidak Ada	
56	1011	Ada	0D
57	1	Tidak Ada	
58	10	Tidak Ada	
59	101	Tidak Ada	
60	1010	Ada	B5
61	1	Tidak Ada	
62	10	Tidak Ada	
63	100	Tidak Ada	
64	1001	Tidak Ada	
65	10011	Tidak Ada	
66	100110	Ada	2E
67	1	Tidak Ada	
68	10	Tidak Ada	
69	101	Tidak Ada	
70	1010	Ada	B5
71	0	Tidak Ada	
72	01	Tidak Ada	
73	011	Tidak Ada	
74	0110	Tidak Ada	
75	01101	Tidak Ada	
76	011011	Ada	35
77	1	Tidak Ada	
78	10	Tidak Ada	
79	100	Tidak Ada	
80	1001	Tidak Ada	
81	10011	Tidak Ada	
82	100111	Ada	0A

Setelah didapat nilai *hexadecimal* dengan teknik *Brute Force*, nilai *hexadecimal* tersebut akan dijadikan kembali menjadi *file* PDF.

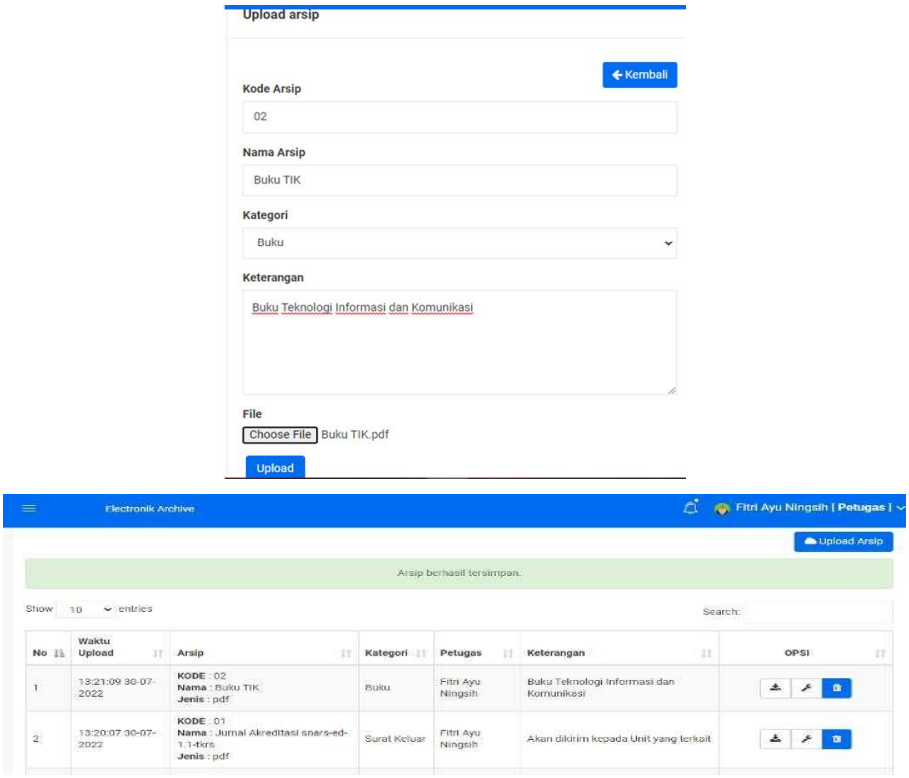


Gambar 5. Sampel *File* PDF Hasil Dekompresi

Jurnal Kajian Ilmiah Teknologi Informasi dan Komputer
ISSN 2962-9055 (Media Online)
Vol 3, No 1, November 2024
Hal 24-33
https://journal.grahamitra.id/index.php/jutik

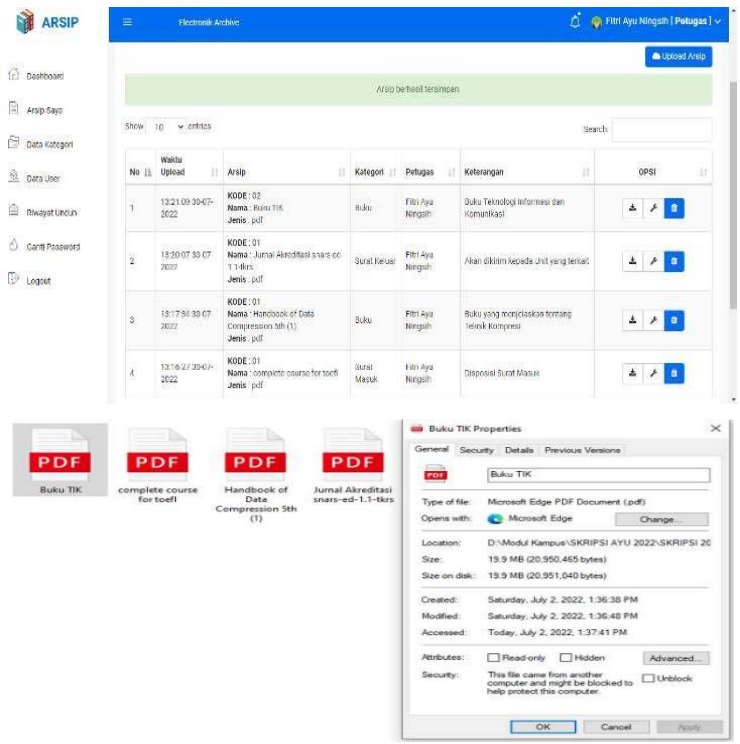
3.2 Implementasi

Penelitian ini terdapat 2 pengujian yaitu:
a. Pengujian berdasarkan fungsionalitas sistem



Gambar 6. Proses Kompresi *File* PDF

Gambar 6 menjelaskan bahwa pada aplikasi *E-Archive* tersedia halaman input data arsip yang dimana halaman tersebut terjadi proses kompresi *file* PDF dengan langkah-langkah mengisi 4 *textfield* berbeda-beda kemudian menekan *button* upload otomatis proses kompresi terjadi ketika data arsip berhasil diupload. Proses kompresi hanya dapat dilakukan oleh petugas/*user*.

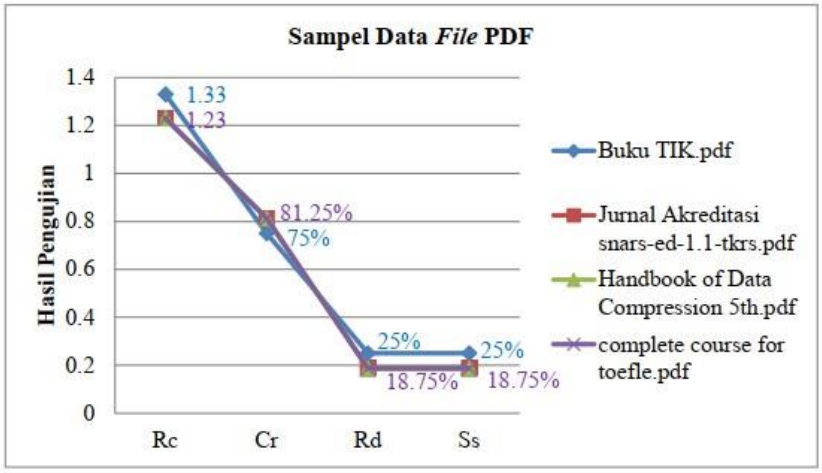


Gambar 7. Proses Dekompresi *File* PDF

Jurnal Kajian Ilmiah Teknologi Informasi dan Komputer
ISSN 2962-9055 (Media Online)
Vol 3, No 1, November 2024
Hal 24-33
https://journal.grahamitra.id/index.php/jutik

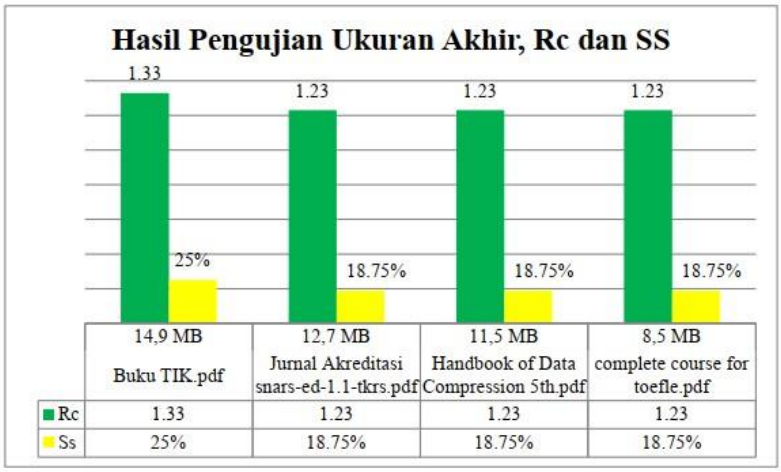
Gambar 7 menjelaskan bahwa pada aplikasi *E-Archive* terdapat halaman lihat data arsip yang dimana halaman tersebut terjadi proses dekompresi *file* PDF dengan cara apabila petugas mengunduh *file* data arsip maka otomatis sistem akan melakukan proses dekompresi. Proses dekompresi dapat dilakukan baik *user* ataupun administrator.

b. Pengujian berdasarkan kualitas kompresi (Rc, Cr, Rd, Ss)



Gambar 8. Grafik Hasil Pengujian

Berdasarkan hasil pengujian pada Gambar 8 dapat disimpulkan bahwa semakin besar ukuran *file* PDF setelah dikompresi maka *Ratio of Compression* (Rc) hasilnya besar, *Compression ratio* (Cr) hasilnya kecil, *Redundancy* (Rd) hasilnya besar, dan *Space saving* (Ss) hasilnya besar, sedangkan semakin kecil ukuran *file* PDF setelah dikompresi maka *Ratio of Compression* (Rc) hasilnya kecil, *Compression ratio* (Cr) hasilnya besar, *Redundancy* (Rd)hasilnya kecil dam *Space saving* (Ss) hasilnya kecil



	14,9 MB Buku TIK.pdf	12,7 MB Jurnal Akreditasi snars-ed-1.1-tkrs.pdf	11,5 MB Handbook of Data Compression 5th.pdf	8,5 MB complete course for toefle.pdf
Rc	1.33	1.23	1.23	1.23
Ss	25%	18.75%	18.75%	18.75%

Gambar 9. Grafik Pengujian Ukuran *File* PDF

Berdasarkan hasil pengujian pada Gambar 9 dapat disimpulkan bahwa *file* PDF yang berjudul buku TIK dengan ukuran setelah dikompresi sebesar 14,9 MB dan nilai parameter *Ratio of Compression* (RC) 1.33, *Space saving* (Ss) 25%, sedangkan *file* PDF yang berjudul *complete course for toefl* dengan ukuran setelah dikompresi sebesar 8,5 MB dan nilai parameter *Ratio of Compression* (RC) 1.23, *Space saving* (Ss) 18.75%, maka *file* PDF yang berjudul *complete course for toefl* memiliki kualitas hasil kompresi terbaik sehingga dapat menghemat ruang penyimpanan pada aplikasi *E-Archive* berbasis *website.*

4. KESIMPULAN

Berdasarkan dari pembahasan penelitian yang telah dilakukan disimpulkan bahwa proses kompresi *file* PDF berlangsung pada saat petugas mengupload *file* PDF yang tersimpan dalam *database* aplikasi *E-Archive* dan proses dekompresi secara otomatis berlangsung pada saat *user* mengunduh *file* PDF yang tersimpan dalam *database* aplikasi *E-Archive*. Proses kompresi menerapkan algoritma *additive code* adalah siapkan *file* PDF yang akan dikompresi kemudian mengkonversi *file* PDF ke nilai *hexadecimal*. Hasil kompresi menjadi *file* PDF dan disimpan di dalam *database* aplikasi *E-Archive.* Dekompresi dilakukan setelah *file* PDF dikompresi. Penerapan algoritma *additive code* menghasilkan bahwa suatu *file*

PDF yang memiliki ukuran yang cukup besar dapat dikompresi menjadi *file* PDF baru dengan ukuran yang lebih kecil sehingga meminimalisir ruang penyimpanan. Hasil pengujian dapat disimpulkan bahwa *file* PDF yang berjudul buku TIK dengan ukuran setelah dikompresi sebesar 14,9 MB dan nilai parameter *Ratio of Compression* (RC) 1.33, *Space saving* (Ss) 25%, sedangkan *file* PDF yang berjudul *complete course for toefl* dengan ukuran setelah dikompresi sebesar 8,5 MB dan nilai parameter *Ratio of Compression* (RC) 1.23, *Space saving* (Ss) 18.75%, maka *file* PDF yang berjudul *complete course for toefl* memiliki kualitas hasil kompresi terbaik sehingga dapat menghemat ruang penyimpanan pada aplikasi *E-Archive* berbasis *website.*

REFERENCES

[1] J. T. Informasi, W. Rahayu, P. S. Informatika, and S. Informasi, "Rancang Bangun Sistem Informasi Akademik Pada SMK Citra Dharma Berbasis JAVA," vol. 5, no. 2, 2019.
[2] S. R. Saragih and D. P. Utomo, "Penerapan Algoritma Prefix Code Dalam Kompresi Data Teks," vol. 4, pp. 249–252, 2020, doi: 10.30865/komik.v4i1.2691.
[3] R. Y. TANJUNG, "Perancangan aplikasi kompresi file dokumen menggunakan algoritma additive code," vol. 8, no. 4, pp. 108–113, 2020, doi: 10.30865/jurikom.v8i4.3593.
[4] B. Tarigan, "Bulletin of Information Technology (BIT) Penerapan Algoritma VLBE Pada Aplikasi Kompresi File," *Bull. Inf. Technol.*, vol. 1, no. 2, pp. 75–82, 2020.
[5] S. B. Ginting *et al.*, "Perbandingan Algoritma Yamamoto ' s Recursive Code Dan Additive Code Dalam Kompresi File Video," vol. 5, 2021, doi: 10.30865/komik.v5i1.3819.
[6] A. N. A. Andrias, "Peranan Sistem Informasi Manajemen Pai Untuk Meningkatkan Kualitas Guru Di Smp Plus Al Munawar," *Textura*, vol. 1,no. 1, pp. 1–12, 2020, [Online]. Available: http://journal.piksi.ac.id/index.php/TEXTURA/article/view/272.
[7] E. Hariska *et al.*, "Perancangan Aplikasi Kompresi File Gambar Menggunakan Algoritma Additive Code," vol. 5, pp. 193–202, 2021, doi: 10.30865/komik.v5i1.3671.
[8] K. Mahesa, "Dekompresi Pada Citra Digital," vol. 12, no. 1, pp. 948–963, 2017.
[9] M. Mawar, "Perancangan Aplikasi Kompresi File Pdf Dengan Menerapkan Algoritma Punctured Elias Codes," *Inf. dan Teknol. Ilm.*, vol. 7, no. 3, pp. 217–223, 2020, [Online]. Available: http://ejurnal.stmik- budidarma.ac.id/index.php/inti/article/view/2391.
[10] R. D. Pratiwi, S. D. Nasution, and F. Fadlina, "Perancangan Aplikasi Kompresi File Teks Dengan Menerapkan Algortima Fixed Length Binary Encoding (Flbe)," *J. Media Inform. Budidarma*, vol. 2, no. 1, pp. 10–14, 2018, doi: 10.30865/mib.v2i1.813.
[11] S. Nainggolan, "Analisa Perbandingan Algoritma Goldbach Codes Dengan Algoritma Dynamic Markov Compression (DMC) Pada Kompresi File Teks Menggunakan Metode Eksponensial," *Maj. Ilm. INTI*, vol. 6, no. 3,pp. 395–399, 2019.
[12] R. A. Sandra, "Implementasi Kombinasi Algoritma Tunstall Code da Boldi-Vigna Untuk Kompresi File Pdf," vol. 8, no. 2, pp. 67–71, 2021.