

Implementasi Kombinasi Algoritma Fibonacci Codes Dan Levenstein Codes Untuk Kompresi File Pdf

Bobby Ramadhana

Teknik Informatika, Fakultas Ilmu Komputer & Teknologi Informasi, Universitas Budi Darma, Medan, Indonesia
Email: bobbyramadhana@gmail.com

Abstrak

Kebutuhan file pdf yang digunakan untuk keperluan pengiriman dengan size yang terlalu besar tentu sangat mempersulit pengiriman, hal ini dikarenakan banyak media pengiriman yang membatasi jumlah size pdf. Masalahnya adalah pada kapasitas penyimpanan yang disediakan, jika file pdf yang terlalu besar dapat mengakibatkan pemborosan ruang penyimpanan dan memperlambat transmisi data. Sehingga dibutuhkan teknik kompresi. Pada penelitian ini akan dilakukan proses kompresi file pdf dengan dua proses algoritma yaitu algoritma Fibonacci Codes dan algoritma Levenstein Codes. Proses kompresi file pdf pertama dilakukan dengan algoritma Fibonacci Codes, kemudian dilanjutkan dengan kompresi kedua menggunakan algoritma Levenstein Codes. Sedangkan untuk proses dekompresi dilakukan sebanyak dua kali yaitu dekompresi pertama menggunakan algoritma Levenstein Codes dan dekompresi kedua menggunakan algoritma Fibonacci Codes. Berdasarkan hasil analisa dengan dua proses kompresi didapatkan space saving kompresi file pdf hingga mencapai 68,75 % sedangkan rasio kompresi sebanyak 31,25%.

Kata Kunci: Kompresi, Fibonacci Codes, Levenstein Codes

1. PENDAHULUAN

Penggunaan pdf biasanya identik dalam dunia kerja yang mengarah pada pengolahan data. Data yang sudah dibuat dalam Microsoft Office seperti word, excel dan lainnya biasa di simpan dalam ekstensi aslinya atau dalam ekstensi lain, salah satunya pdf. Banyaknya orang yang memilih menggunakan format ini adalah karena sangat praktis dan tidak memakan waktu lama untuk membukanya. File pdf juga sering digunakan untuk keperluan pengiriman via email atau lainnya. Banyak perusahaan atau universitas meminta terhadap pengirim file untuk merubah data file kedalam bentuk pdf terlebih dahulu sebelum dikirim. Kebutuhan file pdf yang digunakan untuk keperluan pengiriman dengan size yang terlalu besar tentu sangat mempersulit pengiriman, hal ini dikarenakan banyak media pengiriman yang membatasi jumlah size pdf.

Masalah yang terjadi adalah ukuran data pdf yang besar mengakibatkan pemborosan memori penyimpanan dan memperlambat proses transmisi data. Pemborosan penyimpanan file pdf juga berdampak pada ruang penyimpanan komputer atau media lainnya menjadi penuh. Sehingga akan berakibat penghapusan data-data yang dianggap tidak terlalu memiliki kepentingan. Berdasarkan hal tersebut, dibutuhkan teknik pengecilan data atau biasa disebut dengan teknik kompresi. Kompresi bertujuan untuk mengurangi redundansi data menjadi sekecil mungkin melihat parameter hasil kompresi berupa rasio kompresi dan space saving.

Kompresi data adalah sebuah cara untuk memadatkan data sehingga hanya memerlukan ruangan penyimpanan lebih kecil sehingga lebih efisien dalam menyimpannya atau mempersingkat waktu pertukaran data tersebut. Kompresi data dapat dibagi menjadi dua, lossless compression dan lossy compression. Teknik kompresi memerlukan sebuah algoritma. Ada berbagai macam algoritma Kompresi data salah satunya adalah algoritma Fibonacci Codes dan Algoritma Levenstein Codes Teknik kompresi memerlukan sebuah algoritma. Ada berbagai macam algoritma Kompresi data salah satunya adalah algoritma Fibonacci Codes dan Algoritma Levenstein Codes.

2. TEORITIS

2.1 Kompresi

Kompresi data adalah ilmu atau seni yang merepresentasikan informasi dalam bentuk yang lebih compact [1]. Istilah kompresi tersebut diterjemahkan dari kata bahasa Inggris “compression” yang berarti pemampatan. Dalam bidang teknik, kompresi berarti proses memampatkan sesuatu yang berukuran besar sehingga menjadi kecil. Dengan demikian, kompresi data berarti teknik untuk mengurangi ukuran data agar penyimpanannya jauh lebih padat dan juga untuk mengurangi waktu pengiriman data tersebut [2]. Kompresi data bertujuan untuk mengurangi jumlah bit yang digunakan untuk menyimpan atau mengirimkan informasi.

Tujuan daripada kompresi data tidak lain adalah untuk mengurangi data yang berlebihan (Redundancy Data) sehingga ukuran data menjadi lebih kecil dan lebih ringan sehingga mengurangi biaya untuk penyimpanan [3]. Proses kompresi dan dekompresi data dapat ditunjukkan pada diagram blok.

2.2 Algoritma Fibonacci Codes

Leonardo Pisano Fibonacci adalah seorang ahli matematika Italia, Dia dianggap sebagai Matematikawan terbesar dari abad Pertengahan, Dia berperan penting dalam menghidupkan kembali matematika kuno. Dalam bukunya yang berjudul Liber Abaci, memperkenalkan Eropa dengan notasi Hindu-Arab untuk bilangan [5].

Leonardo Pisano Fibonacci lahir sekitar tahun 1170 dan meninggal sekitar tahun 1250 di Pisa, Italia. Dia menulis teks matematika, antara lain memperkenalkan Eropa dengan notasi Hindu- Arab untuk bilangan. Meskipun buku-bukunya harus ditulis dengan tangan, tetapi teks tentang matematika beredar luas.

Sekarang deret Fibonacci adalah salah satu objek matematika terkenal. Bilangan Fibonacci didefinisikan sebagai barisan bilangan yang suku-sukunya merupakan penjumlahan 2 suku sebelumnya. Bilangan Fibonacci dapat ditunjukkan sebagai barisan bilangan: 0,1,1,2,3,5,8,13,21,34,55,89,144,233,...

2.3 Algoritma Levenstein Codes

Pada tahun 1968 Vladimir Levenstein mengembangkan kode Levenstein yang dapat digunakan untuk bilangan

bulat non-negatif. Proses encoding dan decoding dilakukan di beberapa tahap [6].

Untuk contoh kasus, kita anggap . Nilai biner dari 4 adalah "100", kita ambil "00" dan didapatkan dan maka kode sejauh ini adalah "00". Kemudian hitung nilai biner dari yaitu "10", kita ambil "0" dan didapatkan dan maka kode sejauh ini adalah "000". Kemudian hitung nilai biner yaitu "1", kita ambil "1" dan didapatkan maka kode sejauh ini adalah "000". Karena maka kode adalah tambahkan "1" sejumlah yaitu 3 diikuti "0" dan digabungkan dengan kode sejauh ini sehingga menjadi 1110000. Sehingga 4 pada kode Levenstein adalah 1110|0|00.


3. ANALISA

Pada saat ini banyak orang-orang yang menggunakan file pdf untuk saling bertukar informasi. Contohnya dalam mengirim berkas-berkas yang berhubungan dengan file seperti word teks yang selalu dirubah kedalam file pdf. Banyak perusahaan atau universitas meminta terhadap pengirim file untuk merubah data file kedalam bentuk pdf terlebih dahulu sebelum dikirim. Kebutuhan file pdf yang digunakan untuk keperluan pengiriman dengan size yang terlalu besar tentu sangat mempersulit pengiriman, hal ini dikarenakan banyak media pengiriman yang membatasi jumlah size pdf. Masalahnya adalah pada kapasitas penyimpanan yang disediakan, jika file pdf yang terlalu besar dapat mengakibatkan pemborosan ruang penyimpanan dan memperlambat transmisi data.

Pada penelitian ini akan dilakukan proses kompresi file pdf dengan dua proses algoritma yaitu algoritma Fibonacci Codes dan algoritma Levenstein Codes. Proses kompresi file pdf pertama dilakukan dengan algoritma Fibonacci Codes, kemudian dilanjutkan dengan kompresi kedua menggunakan algoritma Levenstein Codes. Sedangkan untuk proses dekomposisi dilakukan sebanyak dua kali yaitu dekomposisi pertama menggunakan algoritma Levenstein Codes dan dekomposisi kedua menggunakan algoritma Fibonacci Codes, sehingga didapatkan ukuran file pdf awal sebelum terkompresi. Tujuan dari melakukan dua kali kompresi pada file pdf ini adalah untuk mendapatkan hasil berupa parameter kompresi yang lebih maksimal dan optimal seperti space saving dan rasio kompresi.

3.1 Algoritma Levenstein Codes

Dalam melakukan kompresi file pdf, sebelumnya dilakukan analisa file pdf tersebut. Berdasarkan pada batasan masalah, file pdf yang dikompresi berupa file yang berisi karakter string (teks), selain itu tidak bisa. Sebelum melakukan kompresi file pdf, terlebih dahulu menentukan file yang akan dikompresi. Pada contoh kasus ini file pdf yang akan dikompresi diberi nama Sampel.pdf seperti pada gambag di bawah ini:

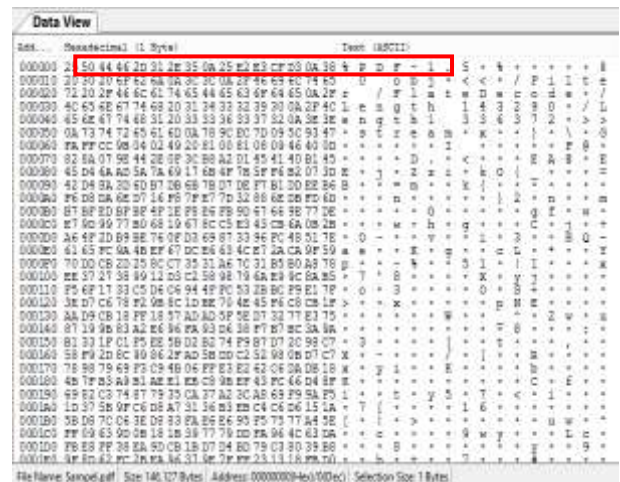
 Sampel.pdf

143 KB

Gambar 1. File PDF Sampel

Berdasarkan pada gambar 1 di atas, file pdf memiliki ukuran 143 KB. Untuk mendapatkan nilai string dari file PDF untuk keperluan hitungan manual maka dibutuhkan aplikasi bantuan. Adapun hasil dari string file

pdf berupa Sampel.pdf dapat dilihat pada gambar di bawah ini.



Gambar 2. Nilai File PDF Sampel

Berdasarkan pada gambar di atas, untuk memudahkan hitungan manual, maka penulis hanya mengambil contoh string file pdf sampel pada baris ke-5 saja yaitu nilai hexadecimal file pdf seperti dibawah ini: String : 65, 6E, 67, 74, 68, 31, 20, 33, 33, 36, 33, 37, 32, 0A, 3E, 3E. Berdasarkan pada string yang akan dijadikan sampel dan contoh kasus pada penelitian ini akan dikompresi menggunakan algoritma Fibonacci Codes terlebih dahulu, kemudian dilanjutkan menggunakan algoritma Levenstein Codes.

a. Kompresi Berdasarkan Algoritma Fibonacci Codes

Untuk memulai proses kompresi, terlebih dahulu setiap string sampel file pdf dirubah kedalam bentuk hexa dan biner kemudian diurutkan dengan memulai dari frekuensi terbesar (banyaknya huruf yang sama) ke yang terkecil. Adapun urutan nilai file pdf sampel yang sudah disusun dapat dilihat pada tabel di bawah ini.

Tabel 1. Urutan Karakter

Hexa	Binner	Bit	Frekuensi	Bit * Frek
33	00110011	8	3	24
3E	00111110	8	2	16
65	01100101	8	1	8
6E	01101110	8	1	8
67	01100111	8	1	8
74	01110100	8	1	8
68	01101000	8	1	8
31	00110001	8	1	8
20	00100000	8	1	8
36	00110110	8	1	8
37	00110111	8	1	8
32	00110010	8	1	8
0A	00001010	8	1	8
Total				128 Bit

Berdasarkan pada tabel 1 di atas, satu nilai hexa/karakter memiliki nilai delapan bit, dan nilai biner didapat dari melihat tabel ASCII. Adapun jumlah nilai hexa file pdf sebanyak 16 nilai dengan total ukuran 136 bit. Setelah nilai bit didapatkan dari semua nilai, selanjutnya adalah melakukan kompresi pertama menggunakan algoritma Fibonacci Codes. Penkompresian dilakukan dengan mengalikan nilai frekuensi dari nilai hexa file pdf yang muncul dengan nilai frekuensi bit dari algoritma Fibonacci Codes. Adapun untuk mendapat nilai bit algoritma Fibonacci

Codes adalah sebagai berikut. a) Tentukan sebuah bilangan bulat positif n yang lebih besar atau sama dengan 1, b) Temukan bilangan Fibonacci f terbesar yang lebih kecil atau sama dengan n , kurangkan nilai n dengan f dan catat sisa pengurangan nilai n dengan f , c) Jika bilangan yang dikurangkan adalah bilangan yang terdapat dalam deret Fibonacci $F(i)$, tambahkan angka "1" pada $i-2$ dalam kode Fibonacci yang akan dibentuk. d) Ulangi langkah 2, tukar nilai n dengan sisa pengurangan nilai n dengan f sampai sisa pengurangan nilai n dengan f adalah 0. e) Tambahkan angka "1" pada posisi paling kanan kode Fibonacci yang akan dibentuk. Adapun tabel nilai fibonacci codes yang didapatkan dari sejumlah banyaknya string file pdf yang akan dikompresi adalah sebagai berikut.

Tabel 2. Kode Nilai Fibonacci

<i>N</i>	<i>Kode Fibonacci</i>
1	11
2	011
3	0011
4	1011
5	00011
6	10011
7	01011
8	000011
9	100011
10	010011
11	001011
12	101011
13	0000011

Selanjutnya dilakukan kompresi string sampel file pdf menggunakan algoritma Fibonacci Codes dengan mengalikan banyaknya frekuensi nilai hexa yang muncul dengan banyaknya bit nilai fibonacci. Adapun proses kompresi tersebut dapat dilihat pada tabel di bawah ini.

Tabel 3. Kompresi Pertama Dengan Nilai Fibonacci

<i>N</i>	<i>Hexa</i>	<i>Fibonacci Codes</i>	<i>Jumlah Bit Fibonacci</i>	<i>Frekuensi Karakter</i>	<i>Bit * Frek</i>
1	33	11	2	3	6
2	3E	011	3	2	6
3	65	0011	4	1	4
4	6E	1011	4	1	4
5	67	00011	5	1	5
6	74	10011	5	1	5
7	68	01011	5	1	5
8	31	000011	6	1	6
9	20	100011	6	1	6
10	36	010011	6	1	6
11	37	001011	6	1	6
12	32	101011	6	1	6
13	0A	0000011	7	1	7
Total					72 bit

Berdasarkan pada tabel 3.3 di atas dapat dibentuk nilai bit baru hasil kompresi dari susunan nilai hexa file pdf sebelum kompresi yaitu "65, 6E, 67, 74, 68, 31, 20, 33, 33, 36, 33, 37, 32, 0A, 3E, 3E" menjadi nilai bit biner. Nilai hexa pertama "65" pada tabel 3.3 kode fibonaccinya adalah 0011, Sehingga untuk hasil seterusnya dapat dilihat pada tabel di bawah ini.

Tabel 4. Susunan Bit Baru Kompresi Pertama

<i>No</i>	<i>Hexa</i>	<i>Kode Fibonacci</i>
1	65	0011
2	6E	1011
3	67	00011
4	74	10011
5	68	01011
6	31	000011
7	20	100011
8	33	11
9	33	11
10	36	010011
11	33	11
12	37	001011
13	32	101011
14	0A	0000011
15	3E	011
16	3E	011
Total		72 bit

Sehingga jika digabung menghasilkan nilai bit baru yaitu:

"001110110001110011010110000111000111111010011110010111010110000011011011".

Proses selanjutnya adalah melakukan penambahan atau padding dan flag bits. Padding dilakukan jika jumlah bit hasil kompresi tidak habis dibagi 8 atau memiliki sisa. Sedangkan Flag bits adalah nilai angka padding yang dijadikan biner. Karena jumlah bit hasil kompresi fibonacci 72 habis dibagi 8 dan tidak memiliki maka tidak perlu ditambahkan padding dan flags bit. Sehingga string bit yang terbentuk keseluruhannya adalah:

"001110110001110011010110000111000111111010011110010111010110000011011011". Adapun total keseluruhan bit adalah 72 bit.

- b. Kompresi Berdasarkan Algoritma Levenstein Codes
Sebelum melakukan kompresi kedua menggunakan algoritma Levenstein, terlebih dahulu hasil kompresi pertama dipisah menjadi 8 bit dan disusun dari frekuensi terbesar hingga terkecil.

Tabel 5. Nilai Hexa dan Binner Hasil Kompresi Pertama

<i>Binner</i>	<i>Hexa</i>
00111011	3B
00011100	1C
11010110	D6
00011100	1C
01111110	7E
10011110	9E
01011101	5D
01100000	60
11011011	DB

Adapun bit hasil kompresi pertama yang telah disusun dapat dilihat pada tabel di bawah ini.

Tabel 6. Urutan Nilai Hasil Kompresi Pertama

<i>Binner</i>	<i>Nilai Hexa</i>	<i>Bit</i>	<i>Frekuensi</i>	<i>Bit * Frek</i>
00011100	1C	8	2	16
00111011	3B	8	1	8
11010110	D6	8	1	8
01111110	7E	8	1	8
10011110	9E	8	1	8
01011101	5D	8	1	8
01100000	60	8	1	8
11011011	DB	8	1	8
Total				72 Bit

Berdasarkan pada tabel 6 di atas, adapun jumlah nilai hexa (binner) sebanyak 9 nilai dengan total ukuran 72 bit. Setelah nilai bit didapatkan dari semua nilai,

selanjutnya adalah melakukan kompresi kedua menggunakan algoritma Levenstein Codes. Penkompresian dilakukan dengan mengalikan nilai frekuensi dari setiap nilai hexa yang muncul dengan nilai frekuensi bit dari algoritma levenstein Codes. Adapun untuk mendapat nilai bit algoritma levenstein Codes adalah sebagai berikut: a) Set angka pertama dari C dengan 1. Letakkan kode-sejauh-ini pada string kosong. b) Ambil nilai biner dari n tanpa angka 1 (buang indeks ke-0 dari nilai biner) di awal dan tambahkan pada kode sejauh ini. c) Nyatakan M sebagai jumlah bit yang ditambahkan pada tahap 2. d) Jika $M \neq 0$, tambahkan C dengan 1 dan lakukan langkah 2 kembali, tetapi dengan nilai M, bukan n. e) Jika $M = 0$, tambahkan 1 sejumlah diikuti dengan 0 ke kode-sejauh-ini dan berhenti Adapun nilai dari Levenstein code yang didapat sesuai dengan jumlah nilai hexa yang dikompresi dapat dilihat pada tabel berikut.

Tabel 7. Kode Levenstein

<i>N</i>	<i>Kode Levenstein</i>
0	0
1	10
2	110 0
3	110 1
4	1110 0 00
5	1110 0 01
6	1110 0 10
7	1110 0 11

Selanjutnya dilakukan kompresi kedua menggunakan algoritma Levenstein Codes dengan mengalikan banyaknya frekuensi nilai hexa yang muncul dengan banyaknya bit nilai Levenstein codes. Adapun proses kompresi tersebut dapat dilihat pada tabel di bawah ini.

Tabel 8. Kompresi Kedua Dengan Nilai Levenstein

<i>N</i>	<i>Hasil Kompresi Pertama</i>		<i>Levenstein Codes</i>	<i>Jumlah Bit Levenstein</i>	<i>Frekuensi Binner</i>	<i>Bit *</i>
	<i>Binner</i>	<i>Hex a</i>				
0	00011100	1C	0	1	2	2
1	00111011	3B	10	2	1	2
2	11010110	D6	110 0	4	1	4
3	01111110	7E	110 1	4	1	4
4	10011110	9E	1110 0 00	7	1	7
5	01011101	5D	1110 0 01	7	1	7
6	01100000	60	1110 0 10	7	1	7
7	11011011	DB	1110 0 11	7	1	7
			Total			40 bit

Berdasarkan pada tabel 8 di atas dapat dibentuk nilai bit baru hasil kompresi dari susunan nilai hexa kompresi pertama yaitu "3B, 1C, D6, 1C, 7E, 9E, 5D, 60, DB" (tanpa tanda koma) menjadi nilai bit biner baru. Misal nilai hexa pertama "3B" pada tabel 3.8 kode levensteinya adalah "10". Sehingga untuk hasil seterusnya dapat dilihat pada tabel di bawah ini.

Tabel 9. Susunan Bit Baru Kompresi Kedua

<i>No</i>	<i>Hexa</i>	<i>Kode Levenstein</i>
1	3B	10
2	1C	0
3	D6	110 0
4	1C	0
5	7E	110 1
6	9E	1110 0 00
7	5D	1110 0 01
8	60	1110 0 10
9	DB	1110 0 11
Total		40 bit

Sehingga jika digabung menghasilkan nilai bit baru yaitu:

"1001100011011110000111000111100101110011".

Proses selanjutnya adalah melakukan penambahan atau padding dan flag bits. Padding dilakukan jika jumlah bit hasil kompresi tidak habis dibagi 8 atau memiliki sisa. Sedangkan Flag bits adalah nilai angka padding yang dijadikan biner. Karena jumlah bit hasil kompresi kedua levenstein 40 juga habis dibagi 8 dan tidak memiliki sisa, maka tidak perlu ditambahkan padding dan flag bits

Sehingga string bit yang terbentuk keseluruhannya adalah:

"1001100011011110000111000111100101110011".

Adapun total keseluruhan bit adalah 40 bit.

Berdasarkan hasil kedua kompresi maka, didapatkan hasil kompresi akhir berupa pengecilan ukuran string sampel file pdf dari 128 bit menjadi 40 bit. Adapun hasil akhirnya setiap bit dipecah menjadi 8 bit dapat dilihat pada tabel di bawah ini.

Tabel 10. Hasil Akhir Kompresi

<i>No</i>	<i>Binner</i>	<i>Nilai Hexa</i>	<i>Bit</i>	<i>Frekuensi</i>	<i>Bit * Frek</i>
1	10011000	98	8	1	8
2	11011110	DE	8	1	8
3	00011100	1C	8	1	8
4	01111001	79	8	1	8
5	01110011	73	8	1	8
			Total		40 Bit

Kemudian hasil kompresi nilai desimal dirubah kedalam bentuk karakter dengan melihat tabel ASCII. Adapun nilai karakter yang dihasilkan dari proses kedua kompresi.

Tabel 11. Karakter Hasil Kompresi

<i>No</i>	<i>Binner</i>	<i>Nilai Hexa</i>	<i>Karakter</i>
1	10011000	98	~
2	11011110	DE	p
3	00011100	1C	FS (file Saparator/ Tidak Tampak)
4	01111001	79	y
5	01110011	73	s

Berdasarkan hasil kompresi dengan *Fibonacci Codes* dan *Levenstein Code* pada proses sebelumnya, adapun dapat dihitung proses kinerja kompresinya berupa parameter yaitu :

1. *Ratio of Compression* (R_c)

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

$$R_c = \frac{128}{40}$$

$$R_c = 5,12$$

2. *Compression Ratio* (C_r)

$$C_r = \frac{\text{Ukuran data setelah dikompresi}}{\text{Ukuran data sebelum dikompresi}} \times 100\%$$

$$Cr = \frac{40}{128} \times 100$$

$$Cr = 31,25 \%$$

3. *Space Saving* (S_s)

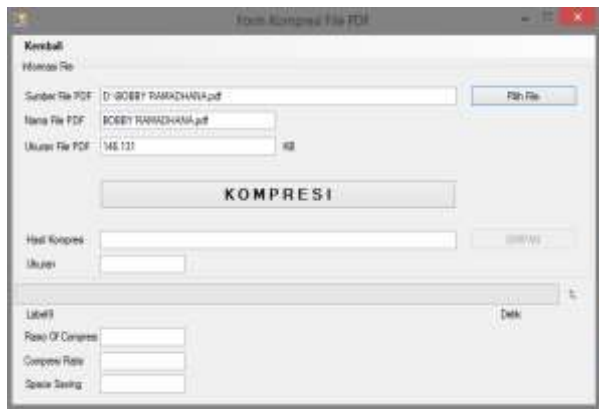
$$S_s = \frac{\text{Ukuran data sebelum dikompresi} - \text{setelah kompresi}}{\text{Ukuran data sebelum dikompresi}} \times 100$$

$$S_s = \frac{128 - 40}{128} \times 100$$

$$S_s = 68,75 \%$$

4. IMPLEMENTASI

User hanya memilih file PDF yang akan dikompresi dan klik button open, sehingga akan tampil informasi file PDF pada textbox seperti gambar di bawah ini.



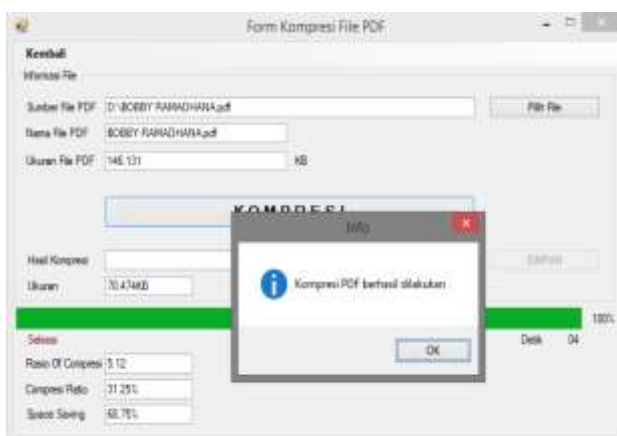
Gambar 3. Informasi file PDF

Berdasarkan pada gambar 1 di atas, untuk memulai proses kompresi user hanya tinggal menekan button KOMPRESI seperti gambar di bawah ini.



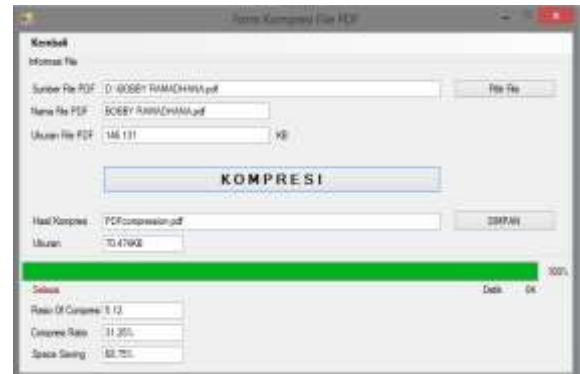
Gambar 4. Proses kompresi fibonacci

Berdasarkan pada gambar 2 proses kompresi kedua dilakukan dengan algoritma Levenstein Code hingga proses selesai seperti gambar di bawah ini.



Gambar 5. Proses kompresi selesai

Berdasarkan pada gambar 3 di atas, klik OK untuk mengakhiri proses kompresi, sehingga didapatkan parameter kompresi dan hasil kompresi file PDF seperti gambar di bawah ini.



Gambar 4. Hasil kompresi file PDF

Adapun proses kompresi file PDF selesai dilakukan, hasil yang didapatkan adalah ukuran baru file PDF hasil kompresi sebesar 70.474 KB, dengan parameter: RC = 5.12
CR = 31.25%
SS = 68.75%

5. KESIMPULAN

Berdasarkan pada gambar 4 di atas, proses kompresi file PDF berhasil dilakukan menggunakan algoritma Fibonacci Codes dan Levenstein codes.

- Proses Kompresi file PDF berhasil dilakukan menggunakan dua algoritma yaitu Fibonacci Codes dan Levenstein codes.
- Pada file PDF hasil kompresi mengalami perubahan ukuran yang signifikan. Rata2 space saving yang didapatkan lebih dari 50%.
- Waktu yang dibutuhkan untuk proses kompresi file PDF bervariasi. Hal ini dipengaruhi oleh panjangnya karakter file yang diinputkan.

Daftar Pustaka

- [1] S. Shanmusugasundaram and R. Lourdasamy, "A Comparative Study Of Text Compression Algorithms. International Journal of Wisdom Based Computing", Vol. 1 (3), pp.68-76, 2011.
- [2] M.A Budiman dan D. Rachmawati, "On Using Goldbach GO Codes and EvenRodeh Codes for Text Compression". Material Science and Engineering 180, 2017
- [3] . Rani and V. Singh, "An Enhanced Text Compression System Based on ASCII Values and Huffman Coding". International Journal of Computer Science Trends and Technology (IJCSST), Vol.4, Issue 3, 2016.
- [4] I.M Pu, Fundamental Data Compression. Jordan Hill, Oxford. 2006
- [5] D. Salomon, Variable-Length Codes for Data Compression. Springer: USA. 2007
- [6] Antoni et al, "Results Analysis of Text Data Compression On Elias Gamma Code, Elias Delta Code and Levenstein Code". International Journal of Science and Advanced Technology, Vol.4, 2014