Penerapan Algoritma LZW (Lempel-Ziv-Welch) pada Program Pemampat File

Verisky Mega Jaya - 13514018
Program Studi Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13514018@std.stei.itb.ac.id

***Abstrak*—Dengan perkembangan teknologi yang begitu cepat, arus informasi digital juga ikut bertambah besar. Data yang dikirim seringkali ukurannya besar sehingga membutuhkan waktu yang lama. Penyimpanan data yang sangat besar juga dapat menimbulkan masalah karena terbatasnya ukuran memori. Masalah ini dapat diatasi dengan memampatkan ukuran data. Makalah ini berisi aplikasi algoritma LZW (Lempel-Ziv-Welch) untuk membuat sebuah program pemampat file yang mangkus. Dengan ukuran kamus yang lebih besar dan kompleksitas pohon yang lebih besar, algoritma ini dapat menghasilkan hasil pemampatan yang lebih baik (lebih kecil) dibandingkan algoritma Huffman yang berbasis algoritma *greedy*.**

Keywords—LZW, Huffman, kamus, pattern matching.

I. PENDAHULUAN

Data yang disimpan pada memori atau yang ditransfer pada jaringan komunikasi pada komputer umumnya berisi informasi yang mubazir (*redundant*) dalam ukuran yang cukup signifikan. Mekanisme atau prosedur mengkode ulang data untuk mengurangi redundansi dapat menggandakan atau bahkan meningkatkan kemangkusan sampai tiga kali lipat.

Beberapa masalah dapat ditemukan saat metode pemampatan yang tidak mangkus terintegrasi dengan sistem computer. Misalnya waktu pemrosesan yang lama karena ukuran data yang besar, tidak dapat memproses beberapa tipe redundansi yang berbeda, dan data hasil kompresi yang ukurannya tidak dapat diprediksi. Pada makalah ini akan dibahas metode pemampatan data LZW (Lempel-Ziv-Welch) yang mempunyai kemampuan lebih yang tidak dimiliki oleh algoritma pemampatan biasa seperti algoritma Huffman.

II. DASAR TEORI

A. Algoritma LZW (Lempel-Ziv-Welch)

Algoritma LZW adalah algoritma pemampatan data yang *lossless* (tanpa kehilangan mutu), diciptakan oleh Terry Welch pada tahun 1984, yang merupakan hasil pengembangan dari algoritma LZ78 yang diciptakan oleh Abraham Lempel dan Jacob Ziv pada tahun 1978.

Tahapan pemampatan data adalah sebagai berikut :

1. Inisialisasi kamus berisi semua string yang panjangnya satu.
2. Cari string terpanjang W didalam kamus yang cocok dengan input yang sedang diproses.
3. Ambil index W sebagai output dan hapus W dari input.
4. Tambahkan W yang digabung dengan simbol selanjutnya ke kamus.
5. Kembali ke tahap 2.

Algoritma ini sebenarnya menerapkan sedikit algoritma greedy saat mencari pola yang cocok dengan yang ada dikamus, Jika pola ditemukan dikamus, maka iterasi akan dilanjutkan terus (menambah ukuran pola) sampai pola tidak ditemukan lagi, sehingga pola yang sebelumnya merupakan pola terpanjang yang mungkin dikodekan.

Berikut adalah contoh memampatkan sebuah teks dengan algoritma LZW.

Teks yang akan di mampatkan :

thisisthe

Tabel proses pemampatan :

No	current	next	output	Add to dictionary
1	T	h	t	th
2	h	i	h	hi
3	i	s	i	is
4	s	i	s	si
5	i	s	-	-

6	is	t	is	ist
7	t	h	-	-
8	th	e	th	the
9	e	-	e	-

Tahap :

1. Karakter gabungan (“th”) belum ada di kamus, sehingga perlu ditambahkan. Karakter current menjadi output.
2. Karakter gabungan (“hi”) belum ada di kamus, sehingga perlu ditambahkan. Karakter current menjadi output.
3. Karakter gabungan (“is”) belum ada di kamus, sehingga perlu ditambahkan. Karakter current menjadi output.
4. Karakter gabungan (“si”) belum ada di kamus, sehingga perlu ditambahkan. Karakter current menjadi output.
5. Karakter gabungan (“is”) sudah ada di kamus. Karakter current menjadi “is”.
6. Karakter gabungan (“ist”) belum ada di kamus, sehingga perlu ditambahkan. Karakter current menjadi output.
7. Karakter gabungan (“th”) sudah ada di kamus. Karakter current menjadi “th”.
8. Karakter gabungan (“the”) belum ada di kamus, sehingga perlu ditambahkan. Karakter current menjadi output.
9. Karakter next sudah tidak ada lagi. Karakter current menjadi output.

B. *Algoritma Huffman*

Algoritma Huffman adalah algoritma pemampatan data yang *lossless* (tanpa kehilangan mutu) yang diciptakan oleh David A. Huffman pada tahun 1952. Pada dasarnya Huffman menerapkan prinsip greedy dalam algoritma ini. Prinsip dari algoritma ini adalah karakter yang sering muncul dikodekan dengan kode yang lebih pendek, sedangkan karakter yang relative jarang dikodekan dengan kode yang lebih panjang.

Tahapan pemampatan data adalah sebagai berikut :

1. Baca semua karakter untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter menyatakan sebuah pohon dengan simpul tunggal yang menyimpan informasi kemunculan karakter tersebut.
2. Gabungkan 2 pohon dengan frekuensi terkecil pada sebuah akar (greedy).
3. Ulangi langkah 2 sampai semua pohon bergabung (tersisa 1 pohon).
4. Baca semua karakter untuk kedua kali untuk proses pengkodean. Jalur dari akar menuju daun yang berisi karakter menyatakan kode Huffman.

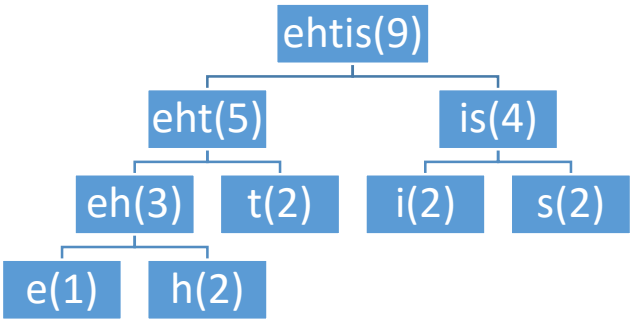
Contoh penerapan algoritma Huffman pada pemampatan teks.

thisisthe

Lakukan perhitungan fekuensi karakter :

No	Karakter	Frekuensi
1	e	1
2	h	2
3	i	2
4	s	2
5	t	2

Bangun sebuah pohon Huffman :



Terakhir, proses mengkodekan karakter dapat dilakukan berdasarkan pohon yang telah dibuat.

C. *Algoritma LZMA2*

Algoritma LZMA adalah algoritma pemampatan data yang lossless (tanpa kehilangan mutu). Algoritma ini dikembangan sejak 1996 dan pertama kali digunakan pada forma file 7z oleh pemampat 7-Zip.

Algoritma ini menggunakan menerapkan algoritma pemampatan dengan kamus yang keluarannya ditentukan dengan *range encoder* menggunakan *complex model* untuk membuat prediksi kemungkinan setiap bitnya.

III. PENERAPAN ALGORITMA LZW PADA PROGRAM PEMAMPATAN DATA

Untuk keperluan percobaan pada makalah ini, program yang berbasis algoritma LZW dibuat menggunakan bahasa

C++. Program menerima masukan sebuah file dan menghasilkan keluaran berupa file hasil pemampatan.

A. *Algoritma utama*

Tahapan pada program LZW ini sebenarnya merupakan pengembangan algoritman Huffman yang sebelumnya, yaitu :

1. Membaca karakter untuk menentukan frekuensi kemunculan. Namun frekuensi kemunculan bukan dihitung per karakter, tapi berdasarkan pola yang tersedia pada kamus.
2. Membentuk pohon bukan berdasarkan karakter, tapi berdasarkan kamus.
3. Membaca karakter kedua kalinya untuk mengkodekan data berdasarkan pohon yang telah dibuat.

```
kamus : array of string
current : string
next : char
combine : string

begin
new file_in
file_in.get(current)
while (file_in masih ada karakter)
        file_in.get(next)
        combine = current + next
        if (combine tidak ada di kamus)
                tambah_ke_kamus(combine)
                count_character(current)
                current = next
        else
                current = combine
endwhile

//membangun tree dari kamus yang sudah
diciptakan
buildtree(kamus)

//Baca untuk kedua kali
new file_in
while (file_in masih ada karakter)
        file_in.get(next)
        combine = current + next
        if (combine tidak ada di kamus)
                tambah_ke_kamus(combine)
                output_code(current)
                current = next
        else
                current = combine
endwhile

end
```

B. *Tipe data pohon*

Sebuah pohon terdiri dari susunan sebuah simpul yang berisi sebagai berikut.

Elemen	Keterangan
Infotype	Terdiri dari Nama dan Frekuensi
*Parent	
*Left	
*Right	

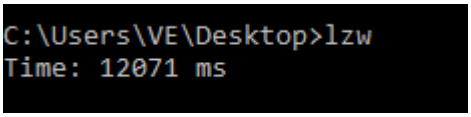
C. *Membangun pohon*

Program LZW menggunakan algoritma yang sama dengan algoritma Huffman untuk membangun pohon. Namun jumlah node yang digunakan lebih banyak dan kompleks karena nodenya bukan hanya terdiri dari maksimal 256 karakter, tapi juga kombinasi-kombinasi dari karakter tersebut yang terdapat di kamus. Pembentukan pohon ini merupakan salah satu yang menyebabkan kompleksitas LZW pada program ini lebih besar dibandingkan program Huffman.

```
while (jumlah elemen kamus > 1)
        left = select_minimum(kamus)
        right = select_minimum(kamus)
        new_root = tree(left, right)
        tambah_ke_kamus(new_root)
endwhile

return new_root
```

D. *Tampilan*

Program hanyalah berupa program berbasis *command prompt* biasa yang menampilkan waktu pemrosesan yang terjadi.

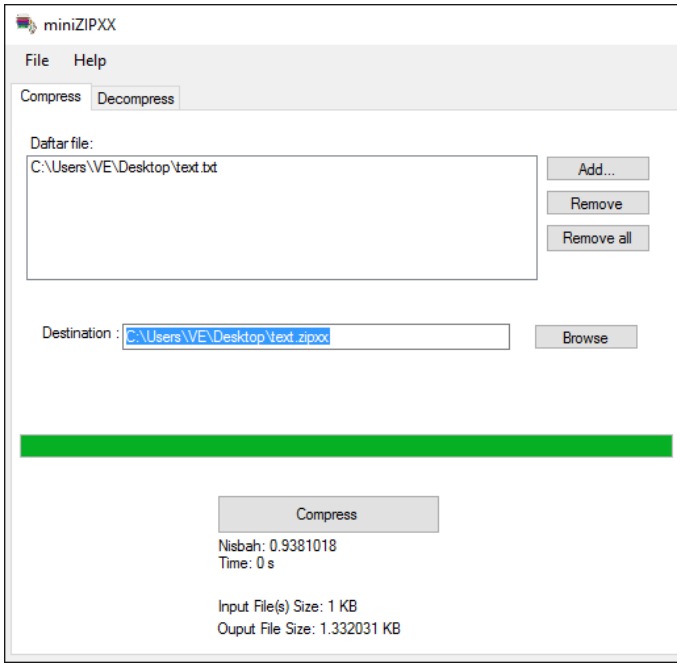


IV. PENGUJIAN DAN PERBANDINGAN

Pada bulan Febuari 2016 lalu peserta mata kuliah IF2211 Strategi Algoritma mendapat sebuah tugas besar untuk membuat sebuah “Mini Winzip” yang merupakan sebuah aplikasi berbasis antarmuka untuk melakukan pemampatan file. Aplikasi dibuat menggunakan bahasa pemrograman C++ dengan bantual kakas yang memungkinkan membuat program berbasis antarmuka seperti Visual Studio. Aplikasi ini menerapkan algoritma *Huffman coding* untuk memampatkan beberapa file sekaligus. Terdapat pula pilihan untuk menggunakan algoritma *adaptive Huffman coding.*

Aplikasi hasil dari tugas besar ini akan menjadi salah satu pembanding yang menggambarkan kerja dari algoritma Huffman. Aplikasi miniZIPXX dibuat dengan kakas Visual Studio 2013 dengan menggunakan algoritma Huffman (*non-adaptive*).

Namun, ukuran file hasil pemampatan dengan menggunakan miniZIPXX kebanyakan tidaklah berbeda jauh dengan ukuran file aslinya. Nisbah hasil pemampatan hanya berkisar 0.9 – 1 , hanya sedikit kasus dimana nisbah bisa berada



dibawah 0.9. Selain itu waktu yang dibutuhkan untuk pemampatan juga cukup lama. Dapat disimpulkan bahwa algoritma ini masih kurang memuaskan.

Dari hasil percobaan tersebut timbulah keinginan untuk mengeksplorasi algoritma lain yang mungkin lebih baik untuk memampatkan file. Salah satu algoritma yang mungkin adalah algoritma LZW (Lempel-Ziv-Welch). Tidak seperti algoritma Huffman yang hanya melihat pola per karakter, LZW juga melihat pola dari gabungan beberapa karakter yang merupakan subset dari data. Gabungan beberapa karakter dapat dikodekan menjadi sebuah kode saja sehingga diharapkan dapat menghasilkan pemampatan yang mangkus.

Selain itu, algoritma LZMA2 juga diikutsertakan sebagai pembanding. Algoritma ini juga dikenal sebagai salah satu algoritma yang sangat mangkus. Salah satu program yang menggunakan algoritma ini adalah 7-Zip.

Pengujian dilakukan beberapa kali hanya untuk 1 file saja setiap kali pengujian karena keterbatasan kemampuan program LZW yang hanya memampatkan 1 file. Perbandingan dilakukan terhadap ukuran file hasil pemampatan dan waktu yang dibutuhkan untuk memampatkan file tersebut. Pengujian hanya dilakukan pada file dengan ukuran sangat kecil (kurang dari 50kB) dikarenakan keterbatasan memori pada program LZW. Program belum dapat membatasi ukuran kamus yang dibuat sehingga file dengan ukuran besar dapat membuat program mengalami *runtime error* karena indeks yang melebihi batas (*index out of range*).

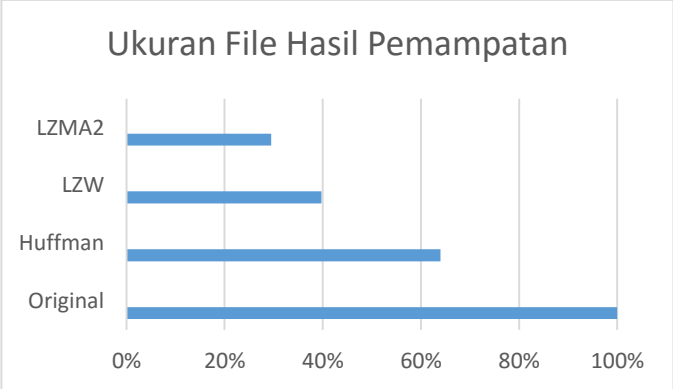
A. Pengujian pada file subtitle (.srt)

File *subtitle* dengan format .srt mempunyai struktur yang berulang-ulang seperti dibawah ini, sehingga pencarian pola dapat dengan mudah dilakukan. Hal ini mengakibatkan file hasil kompresi menjadi sangat kecil jika algoritma yang digunakan dapat menemukan redundansi yang terjadi pada file tersebut.

```
1
00:00:00,182 --> 00:00:03,422
<i>Sebelumnya di The Walking Dead.</i>

2
00:00:03,542 --> 00:00:06,226
Semua! Masuk ke dalam rumah!
Ayo!

3
00:00:06,663 --> 00:00:07,903
Teman-temanku disana.

4
00:00:07,931 --> 00:00:09,664
Istriku hamil.
```

Ukuran file hasil pemampatan :

- File Original : 28,4 kB
- Huffman : 18,3 kB
- LZW : 11,3 kB
- LZMA2 : 8,4 kB



Waktu kompresi :

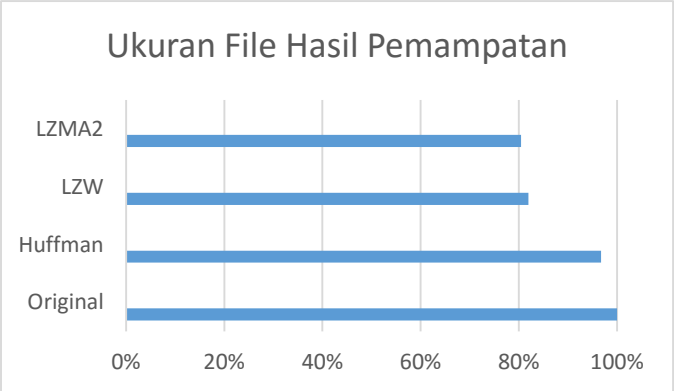
- File Original : -
- Huffman : < 1s
- LZW : 15s
- LZMA2 : < 1s

Hasil pemampatan dengan program LZW dan 7-Zip (LZMA2) sangatlah kecil yaitu kurang dari setengah ukuran file aslinya.

B. Pengujian pada file dokumen (.docx)

File dokumen berekstensi .docx sudah merupakan file hasil pemampatan (Microsoft Word) sehingga jika dimampatkan lagi, hasilnya tidak terlalu signifikan.

Ukuran file hasil pemampatan :
- File Original : 12,8 kB
- Huffman : 12,4 kB
- LZW : 10,5 kB
- LZMA2 : 10,3 kB



Waktu kompresi :

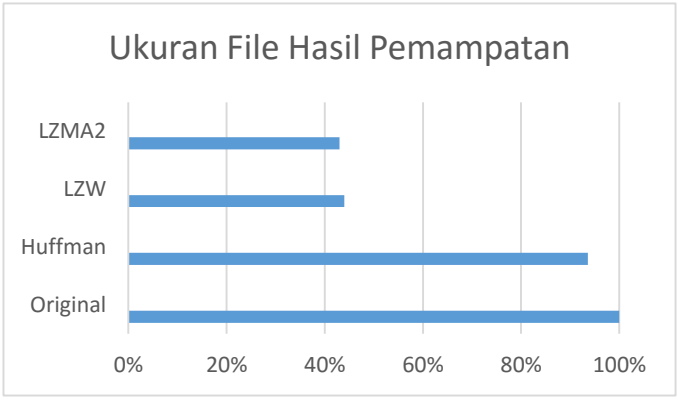
- File Original : -
- Huffman : < 1s
- LZW : 6s
- LZMA2 : < 1s

C. Pengujian pada file teks (.txt)

File teks yang diuji berisi lirik lagu "We Found Love" oleh Rihanna. Pada lirik lagu ini terdapat bagian yang diulang-ulang, sehingga redundansi yang terjadi sangat besar, seperti yang ditunjukan dibawah ini.

```
…
We found love in a hopeless place
We found love in a hopeless place
We found love in a hopeless place
We found love in a hopeless place
```

Ukuran file hasil pemampatan :
- File Original : 1,42 kB
- Huffman : 1,33 kB
- LZW : 0,63 kB
- LZMA2 : 0,61 kB



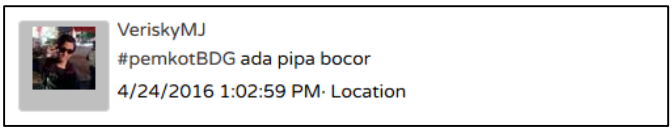
Waktu kompresi :

- File Original : -
- Huffman : < 1s
- LZW : 1.2s
- LZMA2 : < 1s

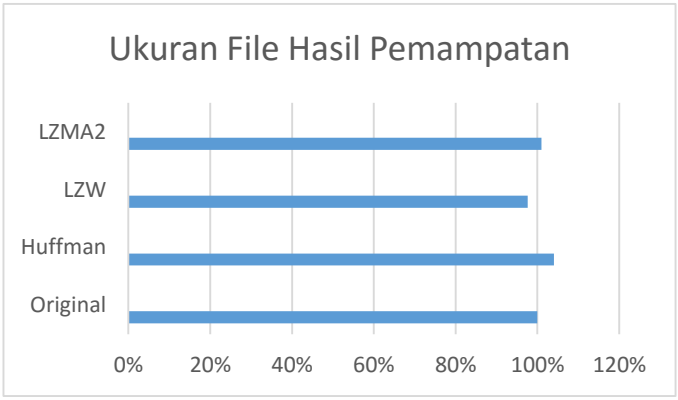
Hasilnya program LZW dan 7-Zip (LZMA2) kembali menghasilkan nisbah yang cukup kecil. Kedua algoritma ini mampu mencari redundansi yang cukup besar.

D. Pengujian pada file gambar (.png)

File gambar berekstensi png sebenarnya sudah merupakan file yang termampatkan, namun pada percobaan ini, dicoba untuk memampatkan lagi sebuah file png.



Ukuran file hasil pemampatan :
- File Original : 17,1 kB
- Huffman : 17,8 kB
- LZW : 16,7 kB
- LZMA2 : 17,2 kB



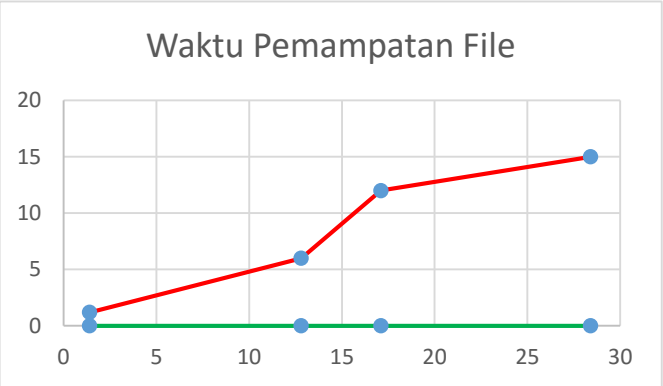
Waktu kompresi :

- File Original : -
- Huffman : < 1s
- LZW : 12s
- LZMA2 : < 1s

Seperti yang diduga sebelumnya, file yang dimampatkan malah bertambah besar untuk sebagian algoritma dikarena format png yang memang sudah mampat.

V. Kemangkusan Aplikasi Algoritma LZW pada Program

Dari hasil beberapa pengujian yang telah dlakukan, dapat disimpulkan bahwa file hasil pemampatan dengan program LZW selalu lebih mangkus dibandingkan dengan program Huffman. Namun hasil yang lebih mampat didapat dengan harga yan besar, yaitu kompleksitas pemrosesan yang lebih banyak, sehingga waktu yang dibutuhkan sangat besar. Diagram menggambarkan hubungan ukuran file (x axis) dan waktu yang diperlukan untuk memampatkan file (y axis). Garis merah menunjukan waktu program LZW dan garis hijau menggambarkan waktu untuk program Huffman.



Terlihat bahwa program LZW yang dibuat masih kurang mangkus dari sisi waktu pemrosesan, namun sudah dapat menghasilkan nisbah yang baik untuk ukuran file hasil pemampatannya.

VI. Kesimpulan

Program berbasis algoritma LZW yang dibuat dapat menghasilkan nisbah pemampatan yang cukup baik, namun sepertinya tahapan yang dilakukan masih kurang baik sehingga waktu pemampatan yang dibutuhkan berkali-kali lipat dibandingkan program berbasis algoritma Huffman. Program ini merupakan pengembangkan lebih lanjut dari program miniZIPXX sebelumnya dengan mengganti algoritma Huffman menjadi algoritma LZW yang lebih kompleks dan menggunakan kamus yang lebih besar. Sehingga hasilnya menambah kompleksitas program tersebut.

Beberapa tahap sepertinya tidak diperlukan untuk mengaplikasikan algoritma ini, seperti membuat pohon Huffman dan membaca file untuk kedua kalinya. Sehinga untuk pengembangan kedepannya masih memungkinkan untuk mengurangi waktu proses pemampatan dengan mengganti kedua proses ini.

Selain itu ukuran kamus yang belum dibatasi membuat program dapat mengalami *runtime error* karena indeks yang melebihi batas jika file yang diproses terlampau besar (yang menghasilkan kamus yang ukurannya besar pula). Pembatasan ukuran kamus dapat dibuat untuk mengatasi masalah ini.

VII. Pengakuan

Puji dan syukur kepada Tuhan Yang Maha Esa yang telah memberikan rahmat-Nya sehingga saya bias menyelesaikan makalah ini. Saya juga mengucapkan terimakasih kepada Dr. Nur Ulfa Maulidevi, S.T., M.Sc dan Dr.Ir. Rinaldi Munir, M.T. yang telah mengajarkan mata kuliah IF2211 Strategi Algoritma yang menjadi pendorong utama dari pembuatan program ini.

References

[1] Welch, Terry (1984). "A technique for high-performance data compression".

[2] Rinaldi Munir, Strategi Algoritma, Institut Teknologi Bandung, 2009.

[3] https://www.youtube.com/watch?v=j2HSd3HCpDs, diakses 7 Mei 2016

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Mei 2016

Verisky Mega Jaya -13514018