Aplikasi “Mini Winzip” untuk Pemampatan dan Penirmampatan *File* dengan Algoritma Huffman (Aplikasi Algoritma Greedy)

Tugas Besar 1

IF2211 Strategi Algoritma

Oleh:

Bervianto Leo P	**13514047**
Ade Surya Ramadhani	**13514049**
Adam Rotal Yuliandaru	**13514091**

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2016

Bab I Deskripsi Tugas

Dalam komunikasi data, pesan yang dikirim seringkali ukurannya sangat besar sehingga waktu pengirimannya lama. Begitu juga dengan penyimpanan data, arsip yang berukuran besar membutuhkan ruang penyimpanan yang besar. Kedua masalah ini dapat diatasi dengan mengkodekan pesan atau isi arsip sesingkat mungkin, sehingga waktu pengiriman pesan relatif cepat dan ruang penyimpanan yang dibutuhkan juga sedikit. Cara pengkodean seperti ini disebut kompresi (pemampatan) data dan pemulihan data tersebut kembali seperti aslinya disebut dekompresi (penirmampatan). Salah satu cara pemampatan dan nirmampat data ini adalah dengan menggunakan kode Huffman.

Pada tugas pertama Strategi Algoritma ini, anda diminta membuat aplikasi "*mini WinZip*" yang mengimplementasikan algoritma Huffman untuk memampatkan dan menirmampatkan data. Algoritma Huffman menggunakan prinsip *greedy* dalam pembentukan kode Huffman. Program anda harus dapat memampatkan semua jenis data baik berupa teks, gambar, suara, dan video dan Anda harus mampu mengembalikan data yang sudah dikompres tersebut ke bentuk asalnya (dekompresi). Sebagai contoh, jika *executable file* dimampatkan (misalnya `notepad.exe`), maka program `notepade.exe` tersebut harus dapat dijalankan kembali.

Program yang Anda buat selain mampu memampatkan dan menirmampatkan data juga harus dapat menunjukkan perubahan hasil kompresi tersebut melalui nisbah pemampatannya. Nisbah (*ratio*) pemampatan dhitung dengan rumus:

P = Ukuran *file* hasil pemampatan/Ukuran *file* sebelum dimampatkan $\times$ 100%

yang berarti ukuran *file* menjadi *P* (dalam persentase) dari ukuran semula.

Waktu yang dibutuhkan untuk memampatkan dan menirmampatkan data juga dicatat. Selain itu, program harus dapat memampatkan banyak *file* sekaligus, dan ketika penirmampatan harus dapat mengembalikannya menjadi *file-file* semula dengan nama sama. Hitung juga entropi pesan dan rata-rata panjang bit setiap simbol di dalam pesan (file). Lihat Lampiran untuk menghitung entropi.

Extension: Algoritma Huffman standard memampatkan data agak lambat karena file harus dibaca dua kali. Pertama, file dibaca untuk menghitung frekuensi kemunculan karakter di dalam file. Kedua, file dibaca kembali untuk mengkodekan setiap karakter di dalam file dengan kode Huffman. Varian dari algoritma Huffman adalah algoritma *adaptive Huffman coding*. Pada algoritma adaptif ini, file dibaca hanya satu kali saja. Setiap kali karakter dibaca, pohon Huffman dibangun dan kode Huffman untuk karakter itu ditentukan, sekaligus karakter dikodekan langsung.

Spesifikasi program :

1. Program mampu memampatkan berkas (*file*) berjenis apapun secara tepat dan benar.
2. Program mampu memampatkan banyak file sekaligus menjadi satu berkas mampat.
3. Program mampu menirmampatkan data yang sebelumnya telah dikompres dengan tepat dan benar.
4. Program harus mampu menampilkan proses pemampatan dan nirmampat tersebut (misal dalam *progressive bar*).
5. Program juga harus manampilkan statistik berupa: lama waktu pemampatan (dan penirmampatan), ukuran file semula, ukuran file setelah dimampatkan, nisbah pemampatan, entropi, rata-rata bit/simbol, dll.
6. Antarmuka program sebaiknya kompatibel seperti *Winzip*.

Bab II Dasar Teori

Algoritma *Greedy* merupakan metode yang paling populer untuk memecahkan persoalan optimasi. Greedy sendiri diambil dari bahasa inggris yang artinya rakus, tamak atau serakah. Prinsip algoritma greedy adalah: “take what you can get now!”. [1]

Algoritma *Huffman* adalah salah satu algoritma kompresi. Algoritma *Huffman* ini merupakan salah satu algoritma *Greedy*. Terdapat tiga fase dalam menggunakan algoritma *Huffman* untuk mengompres sebuah teks, pertama adalah fase pembentukan pohon *Huffman*, kedua fase *encoding* dan ketiga fase *decoding*. Prinsip yang digunakan oleh algoritma *Huffman* adalah karakter yang sering muncul di-*encoding* dengan rangkaian bit yang pendek dan karakter yang jarang muncul di-*encoding* dengan rangkaian bit yang lebih panjang. Teknik kompresi algoritma *Huffman* mampu memberikan penghematan pemakaian memori sampai 30%. Algoritma Huffman mempunyai kompleksitas $O(n \log n)$ untuk himpunan dengan n karakter. [2]

Pohon adalah struktur data yang tidak linier/non linier yang digunakan terutama untuk mempresentasikan hunbungan data yang bersifat hiererkis antara elemen – elemennya. Elemen dari tree sendiri terdiri dari root (akar) dan sisa elemen yang lain disebut simpul (node/vortex) yang terpecah menjadi sejumlah himpunan yang tidak saling berhubungan satu sama lain (subtree/cabang). **Pohon biner** sendiri merupukan varian dari tree yang elemen cabang nodenya memiliki maksimal 2 subtree. [4]

Bab III Analisis Pemecahan Masalah

Langkah – langkah kompresi file

- Membaca File dalam bentuk byte/per karakter
- Membuat tabel frekuensi untuk semua char yang pernah muncul
- Sesuai dari tabel tersebut diurutkan berdasarkan kemunculan yang paling kecil ke besar (kami menggunakan priority Queue of Tree)
- Membuat pohon huffman dengan mengambil 2 char dengan frekuensi terkecil dan membuat *tree* baru kemudian di add kembali ke Queue. Lakukan langkah tersbut sampai terbentuk pohon Huffman
- Pengaksesan kembali file tersebut dan membentuk encode ke file compress berdasarkan pohon huffman yang sudah ada
- File compress hasil dituliskan juga nama file asal, tabel frekuensi kemunculan dan hasil encode file
- Jika file lebih dari satu ulang langkah –langkah tersebut sampai semua file berhasil dicompress

Langkah - langkah dekompresi file

- Membaca file dalam bentuk byte/per char
- membaca nama file yang akan didekompres
- Membaca tabel frekuensi karakterter dan membuat pohon huffman berdasarkan tabel tersebut
- Mendecode kode biner file ke pohon huffman yang sudah ada
- Buat file baru dan tuliskan hasil decode disana
- Jika file lebih dari satu maka langkah-langkah tersebut diulang sampai end of file

Struktur data yang kami gunakan adalah class Queue dan Tree. Kami juga menggunakan beberapa *library* bawaan C++ seperti *iostream*, *fstream*, *string.h*, *cctype*, *stdlib.h*, *math.h*. Lalu untuk *Graphical User Interface* kami menggunakan kakas Qt Creator.

Spesifikasi program

- Program mampu memampatkan berkas (*file*) berjenis apapun secara tepat dan benar.
- Program mampu memampatkan banyak file sekaligus menjadi satu berkas mampat.
- Program mampu menirmampatkan data yang sebelumnya telah dikompres dengan tepat dan benar.
- Program harus mampu menampilkan proses pemampatan dan nirmampat tersebut (misal dalam *progressive bar*).
- Program juga harus manampilkan statistik berupa: lama waktu pemampatan (dan penirmampatan), ukuran file semula, ukuran file setelah dimampatkan, nisbah pemampatan, entropi, rata-rata bit/simbol, dll.

Bab IV Implementasi dan Pengujian

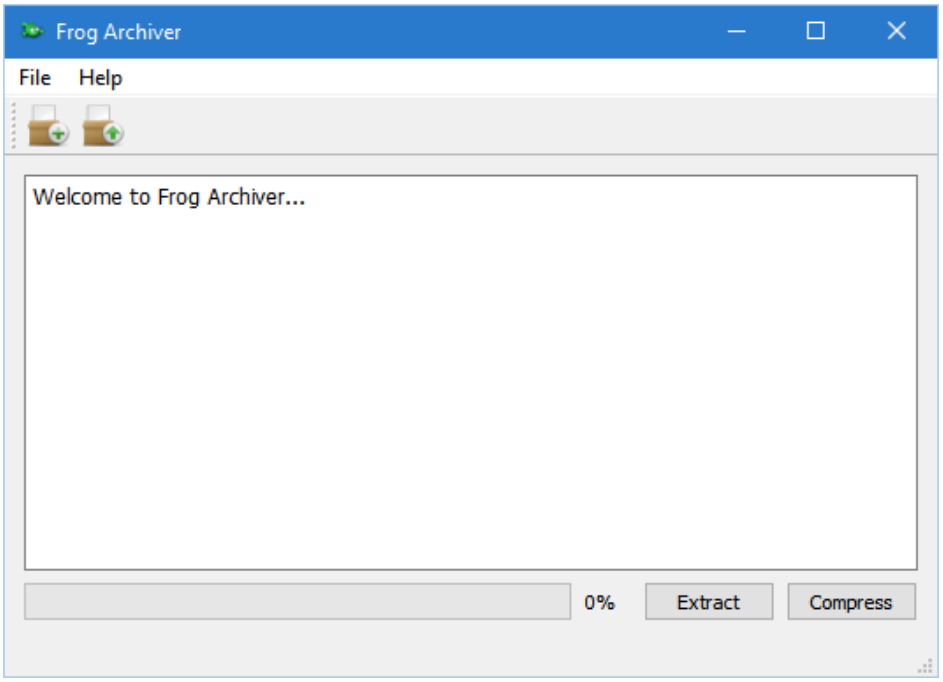
4.1 Implementasi

Kakas yang digunakan yaitu Qt Creator sehingga akan menggunakan library Qt sebagai antarmuka. Antarmuka dan interaksinya terdapat pada *mainwindow.cpp* sedangkan desain dari antarmuka tersebut terdapat dalam *mainwindow.ui,* antarmuka ini merupakan jendela awal sebagai interaksi dengan pengguna yang merupakan window utama atau sebagai kelas *mainwindow*. Kelas *mainwindow* juga merupakan pengatur jalannya aplikasi. Selain itu terdapat jendela tambahan lain seperti *about* yang berisi informasi mengenai aplikasi beserta pembuat aplikasi. Jendela atau kelas *about* diimplementasikan dalam file *about.h* sebagai pendefinisian kelas *about*, *about.cpp* sebagai implementasi dari *about.h* dan *about.ui* sebagai desainnya. Beberapa kelas ini merupakan kelas yang mengatur antarmuka dan interaksi dengan pengguna.

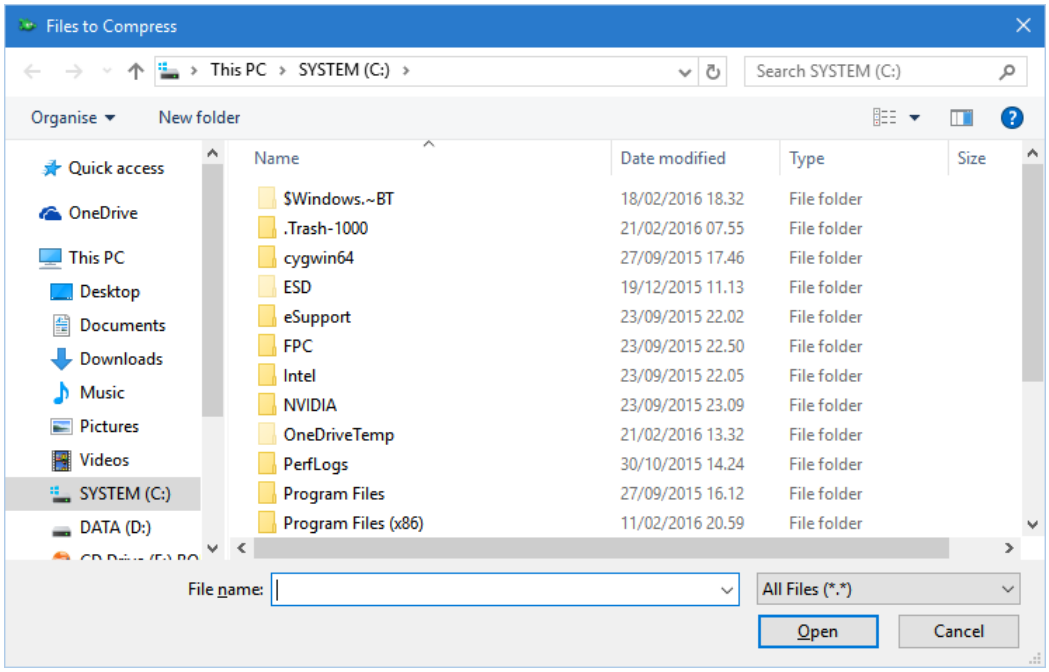
Selain itu terdapat kelas pohon, *encoder*, dan *decoder*. Kelas huffman diimplementasikan dalam *huffman.h* . Kelas *huffman* merupakan kelas yang mengatur struktur data sebagai penyimpanan dalam *encoder* maupun *decoder*. Didalam kelas ini terdapat *encoder* dan *decoder. Encoder* memiliki peran membaca *file* untuk siap di-*encode* dan memampatkan beberapa *file* menjadi satu *file* yang sudah dimampatkan. Terakhir yaitu *decoder* yang merupakan penerjemah dari *file* yang sudah dimampatkan sehingga menjadi beberapa atau satu *file* yang sebelumnya telah dimampatkan.

Berikut ini beberapa screenshot antarmuka program :

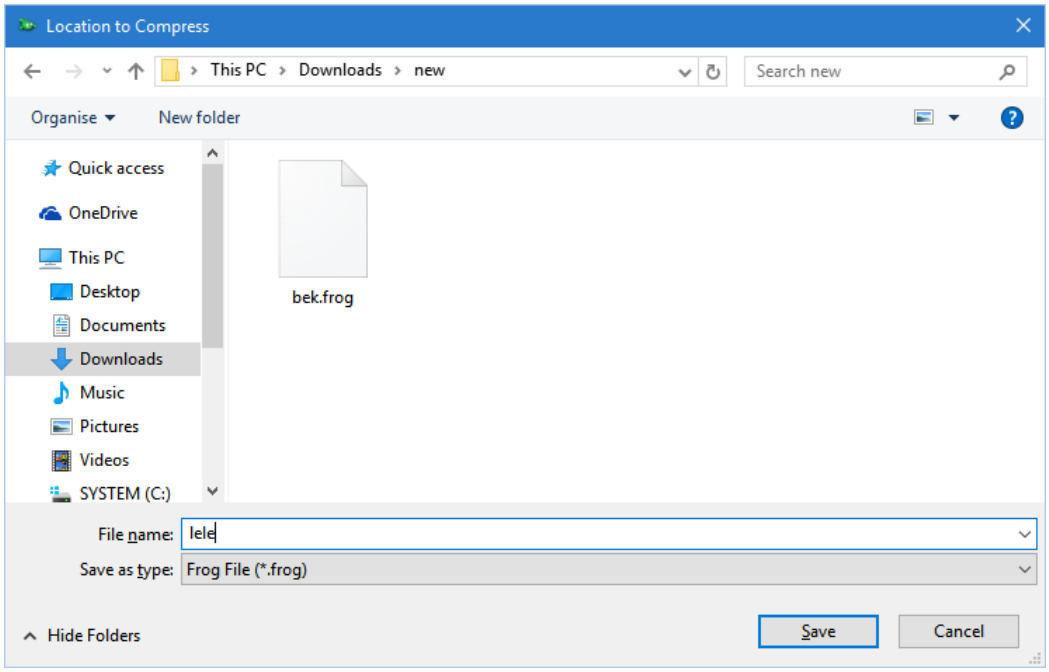
a. Antarmuka utama



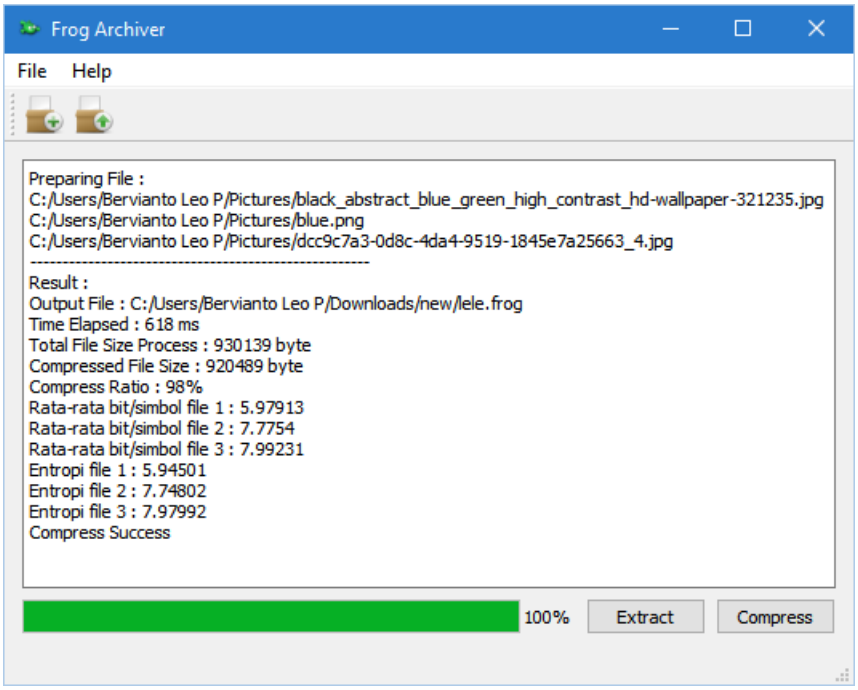
b. Dialog membuka *file* untuk dimampatkan



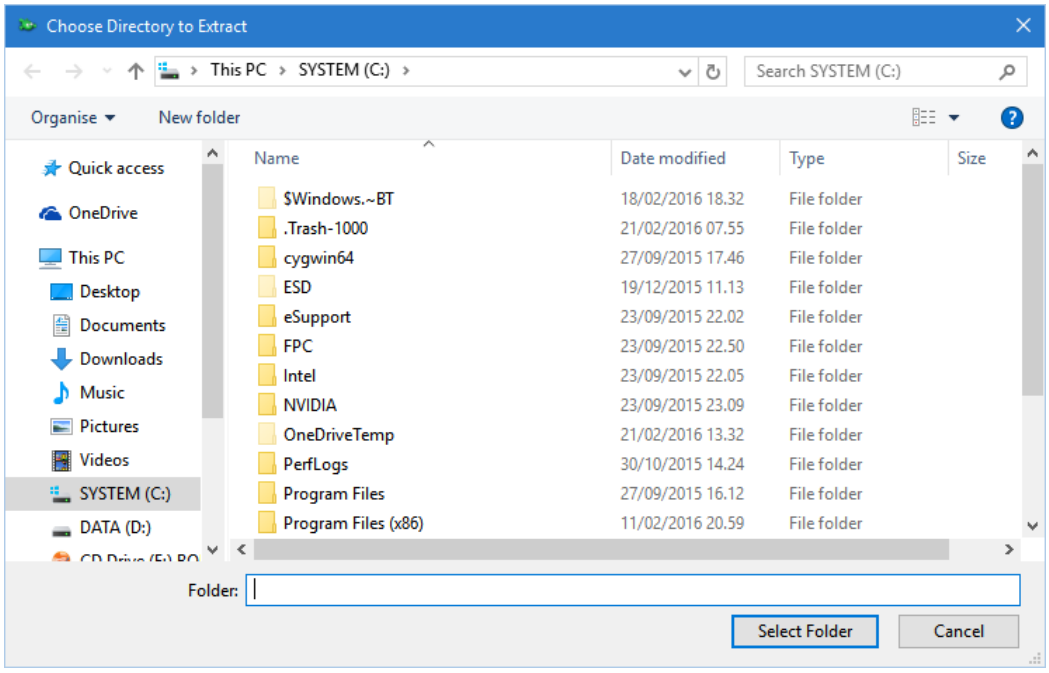
c. Dialog menyimpan nama *file* dan lokasi untuk memampatkan



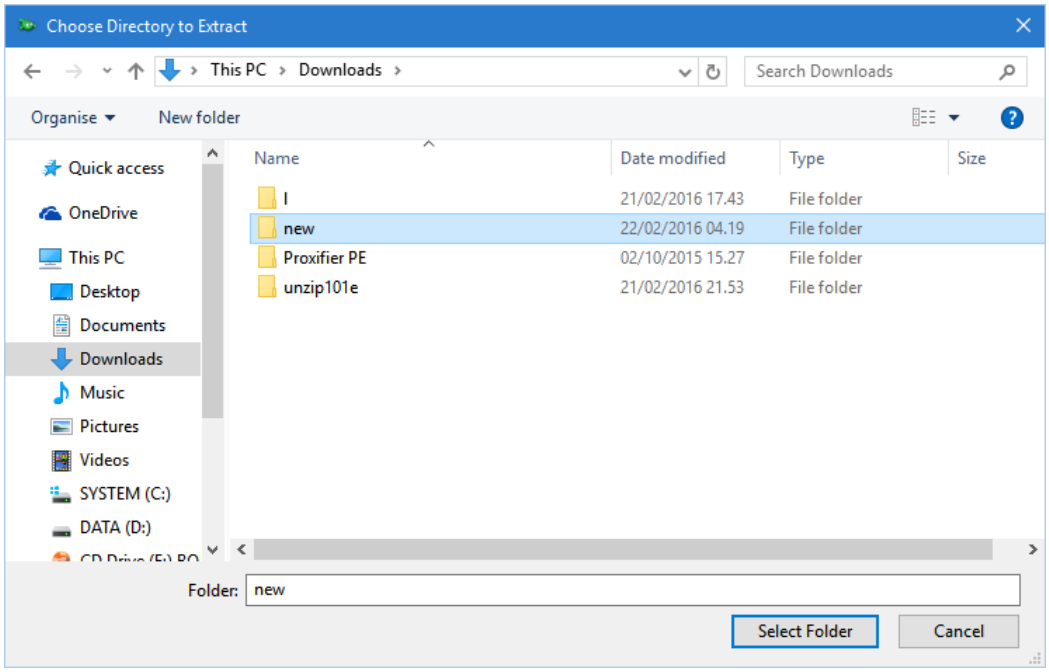
d. Perubahan antarmuka utama saat memampatkan



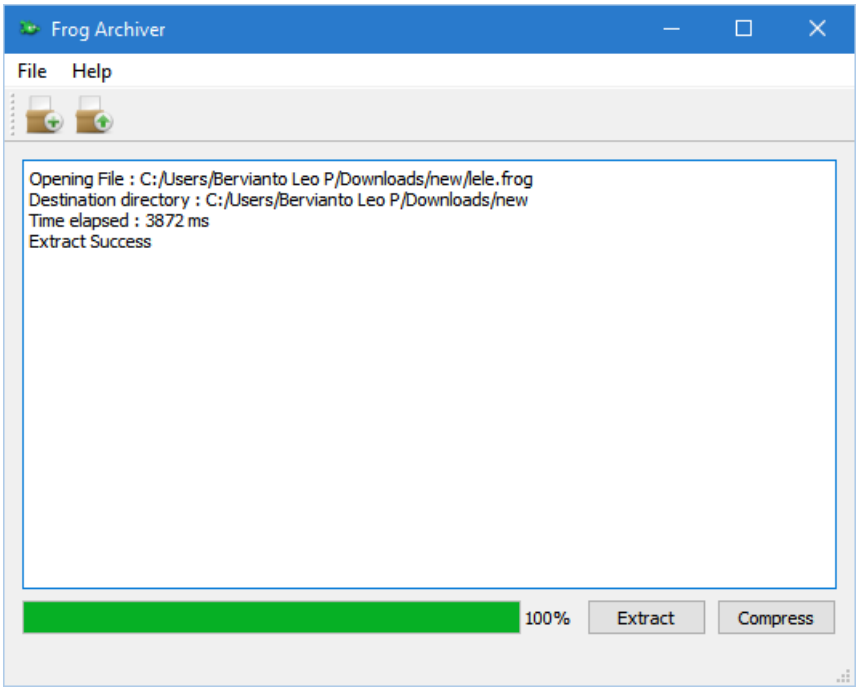
e. Dialog membuka *file* yang telah dimampatkan



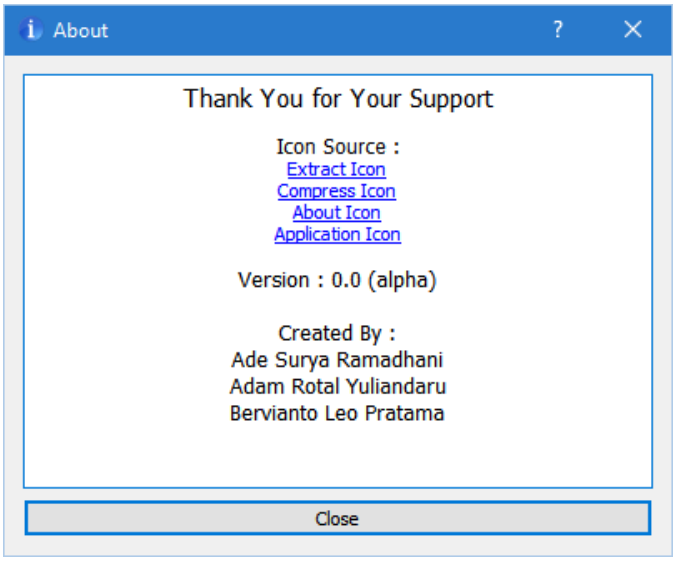
f. Dialog memilih direktori untuk menirmampatkan



g. Perubahan antarmuka utama saat menirmamptkan



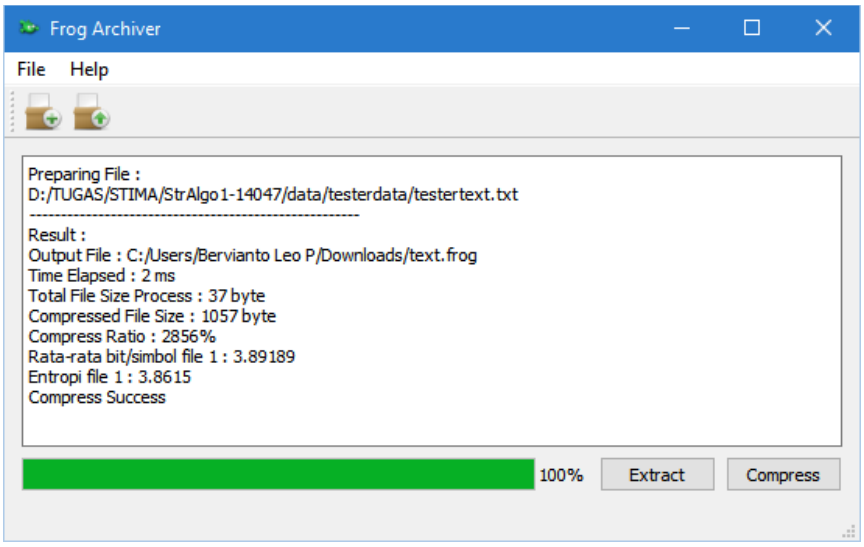
h. Antarmuka tambahan (tentang program)



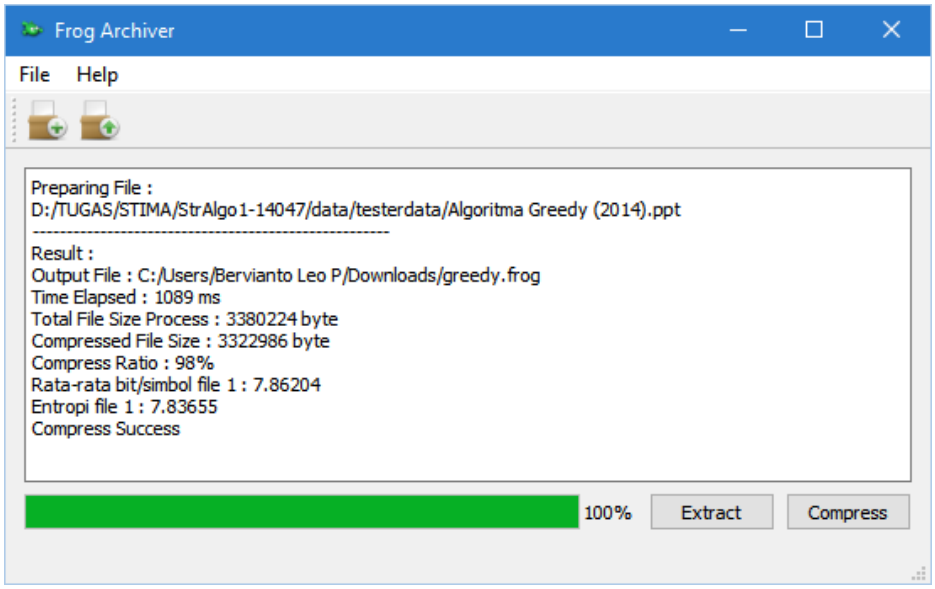
4.2 Pengujian

Berikut ini beberapa hasil pengujian yang dilakukan :

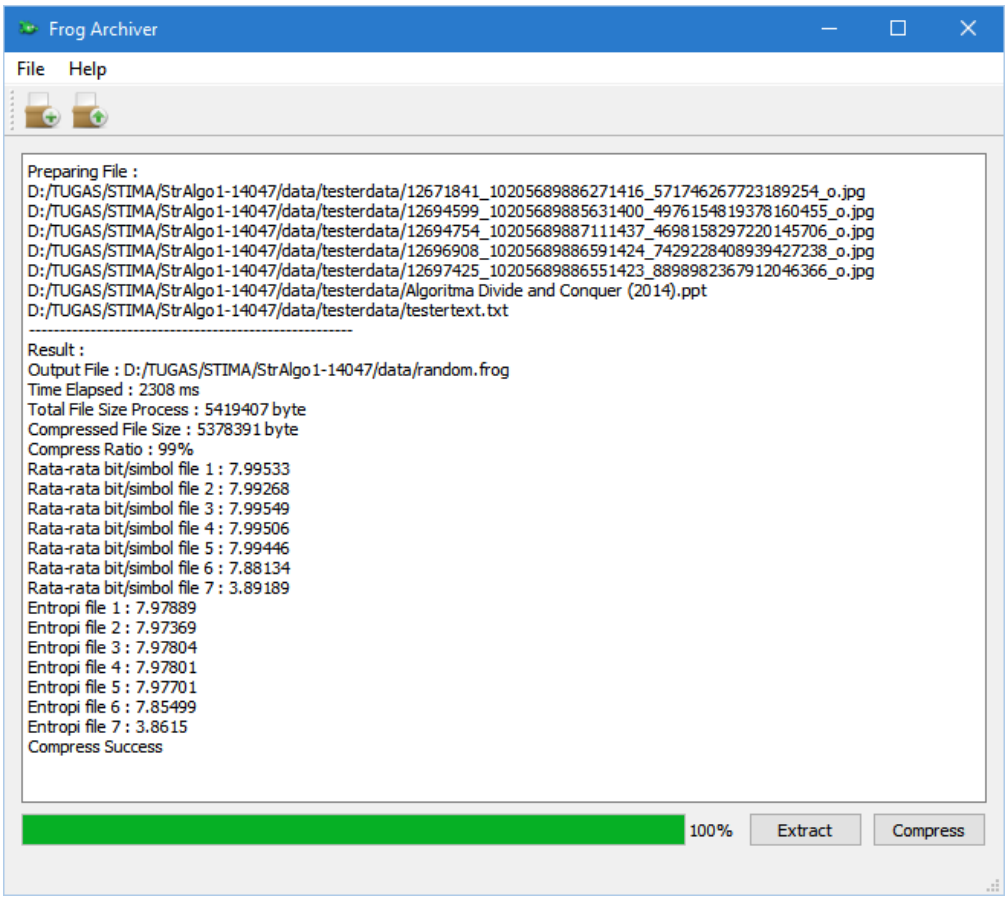
a. Pemampatan pada file teks berukuran kecil



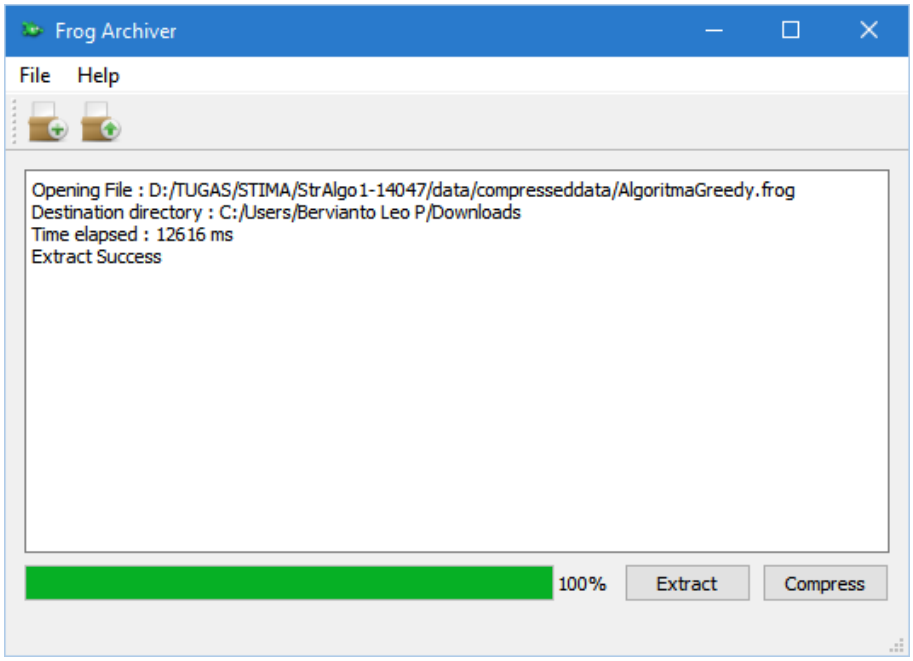
b. Pemampatan pada file bebas berukuran besar



c. Pemampatan pada banyak file yang beragam



d. Penirmampatan file yang memiliki satu file besar



Analisis hasil pengujian :

Pada pemampatan file kecil akan memberikan file kompresi yang melebihi file aslinya, dikarenakan header yang disimpan lebih besar dari isinya. Akan lebih efektif pada file yang cukup besar seperti percobaan b begitu juga c dengan beberapa file yang cukup banyak dan besarnya beragam. Dari segi waktu eksekusi akan berbeda dan akan lebih lama pada penirmampatan.

Bab V Simpulan dan Saran

5.1 Simpulan

Berdasarkan proses pembuatan kompresi dan dekompresi menggunakan algoritma huffman ini, maka didapatkan beberapa kesimpulan antara lain:

- Algoritma Huffman dapat diterapkan untuk melakukan kompresi dan dekompresi multi-file.
- Algoritma Huffman dapat dikategorikan cepat dalam melakukan kompresi dan dekompresi dengan berbagai ukuran file.
- Semakin besar ukuran suatu file, maka lebih efektif dalam memampatkan data.

5.2 Saran

- File yang dikompres menggunakan algoritma huffman sebaiknya berukuran cukup besar, sehingga lebih efektif.

Daftar Pustaka

[1] http://www.it-jurnal.com/2015/09/pengertian-algoritma-greedy.html) (diakses pada tanggal 18 Februari 2016, pukul 11.45)

[2] http://marcoturnip.blog.widyatama.ac.id/2014/06/02/kompresi-teks-menggunakan-algoritma-huffman/ (diakses pada tanggal 18 Februari 2016, pukul 11.30)

[3] Dokumentasi Qt - http://doc.qt.io/

[4] http://dhedee29.staff.gunadarma.ac.id (diakses 21 februari 2016 pukul 17.30)

[5] http://code.activestate.com/recipes/577480-huffman-data-compression/ (diakses tanggal 18 februari pukul 20.00)