BAB III

METODOLOGI DAN PERANCANGAN SISTEM

3.1 Metodologi Penelitian

Metodologi penelitian yang akan digunakan dalam penelitian ini antara lain adalah sebagai berikut:

1. Wawancara

   Wawancara dilakukan untuk mengetahui informasi mengenai *E-learning* UMN. Selain itu juga untuk mengetahui pendapat dari kepala *E-learning* tentang data yang tersimpan di *server E-learning* UMN.

2. Studi Literatur

   Dalam studi literatur dilakukan studi mengenai teori-teori, konsep, serta penelitian sebelumnya yang berkaitan dengan pokok bahasan penelitian, antara lain mengenai kompresi data, algoritma LZ77, algoritma Huffman, algoritma Brotli, *E-learning*, dan aplikasi Moodle. Sumber-sumber yang digunakan dapat berupa buku, jurnal ilmiah, artikel, dan lain-lain yang terdapat pada media cetak maupun media internet.

3. Analisis dan Perancangan

   Analisis dilakukan dari hasil wawancara dan hasil pemahaman berdasarkan studi literatur sekaligus perancangan terhadap sistem kompresi yang akan diimplementasikan ke dalam *E-learning* UMN. Sistem kompresi dibuat dengan bahasa pemrograman Java dengan membuat *plugin* pada Moodle.

4. Implementasi

   Proses implementasi dilakukan dengan membangun sistem kompresi dengan bahasa pemrograman Java dari hasil rancangan yang telah didefinisikan. Plugin pada *e-learning* dibangun menggunakan bahasa pemrograman PHP. Proses pengimplementasian dilakukan pada dua tahap, tahap pertama dilakukan pada server Moodle lokal dan tahap kedua akan di implementasi langsung ke dalam Moodle milik *E-learning* UMN.

5. Pengujian dan Evaluasi

   Pengujian dilakukan terhadap algoritma Brotli yang dipakai dan evaluasi terhadap *file* dokumen dan gambar yang dikompresi menggunakan algoritma Brotli dan disimpan ke dalam *server E-learning* UMN. Pengujian dilakukan dengan membandingkan aplikasi yang dibangun dengan aplikasi milik Google. Evaluasi dilakukan dengan menghitung rasio kompresi, faktor kompresi, dan persentasi penghematan terhadap *file* hasil kompresi.

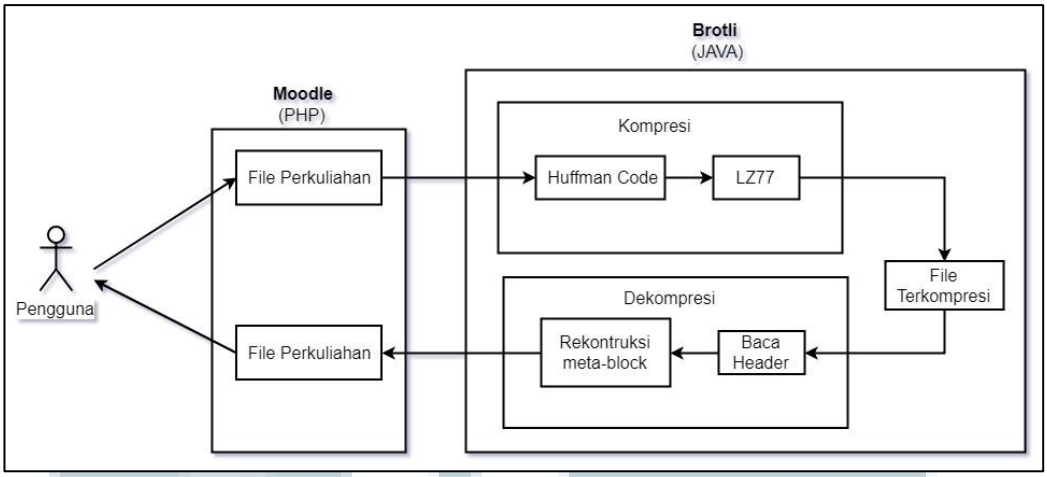
6. Dokumentasi

   Dokumentasi dilakukan dari awal penelitian hingga akhir penelitian pada seluruh proses yang terjadi dalam penelitian ini. Dokumentasi dimulai dari perancangan *plugin* hingga perumusan kesimpulan dari penelitian.

3.2 Model Aplikasi

Dalam proses kompresi, sistem membutuhkan masukkan berupa *file* yang akan dikompresi. Setelah itu, sistem akan melakukan proses kompresi. Pada proses kompresi, *file* akan dikompresi menggunakan algoritma Huffman dengan konteks

pemodelan kedua dan dilanjutkan dengan algoritma LZ77. *Output* dalam proses ini adalah *file* yang sudah terkompresi.



Gambar 3.1 Model Aplikasi

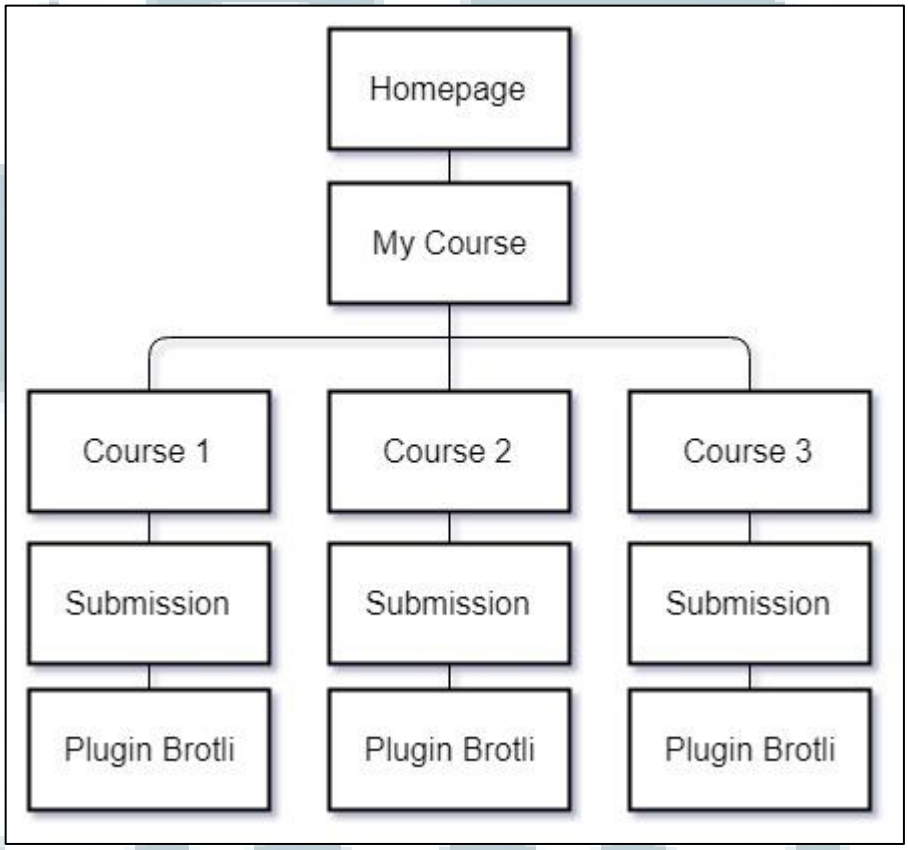
Sedangkan dalam proses dekompresi, sistem akan membaca ukuran *sliding window* dari *header* pada *file* kompresi dan melakukan dekompresi terhadap setiap *meta-block* pada *file* kompresi. *Output* dalam proses ini adalah *file* hasil dekompresi.

3.3 Perancangan

Aplikasi yang dirancang mengimplementasikan teknologi kompresi data menggunakan algoritma Brotli pada server *E-learning* UMN. Aplikasi akan dibuat dalam bentuk *plugin* untuk Moodle. Aplikasi *plugin* dirancang agar dapat melakukan proses kompresi dan dekompresi terhadap setiap *file* yang diunggah. Aplikasi dirancang menggunakan mode UTF-8 dimana setiap *file* masukkan akan diubah ke format UTF-8 sebelum dilakukannya proses kompresi.

Aplikasi *plugin* akan menerima *file* yang diunggah pengguna dan melakukan kompresi sebelum akan dipindahkan ke dalam *file* sistem Moodle. Jika proses unggah selesai *plugin* akan menampilkan pesan bahwa *file* berhasil

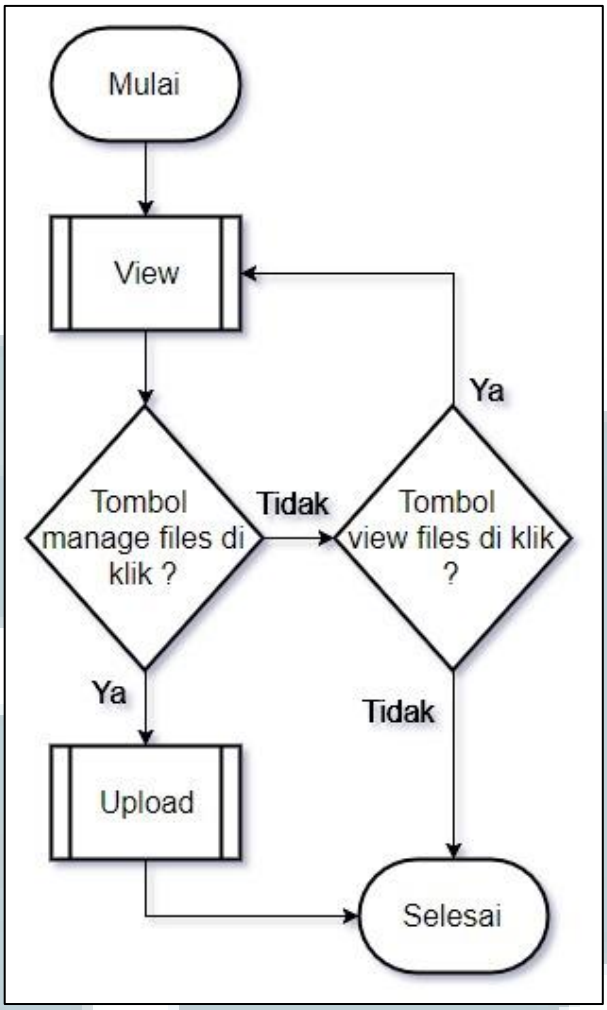
diunggah. Dari sana pengguna dapat berpindah halaman untuk melihat daftar *file* yang sudah terunggah dan pengguna dapat mengunduh *file* tersebut. Pada halaman pengunduhan, *file* akan didekompresi terlebih dahulu sebelum menampilkan *link* pengunduhan pada halaman tersebut. *Sitemap* dari pada *plugin* akan dijabarkan pada Gambar 3.2.



Gambar 3.2 *Sitemap plugin E-learning*

3.3.1 Perancangan Aplikasi

Rancangan aplikasi *plugin* secara keseluruhan dapat dilihat pada Gambar 3.3

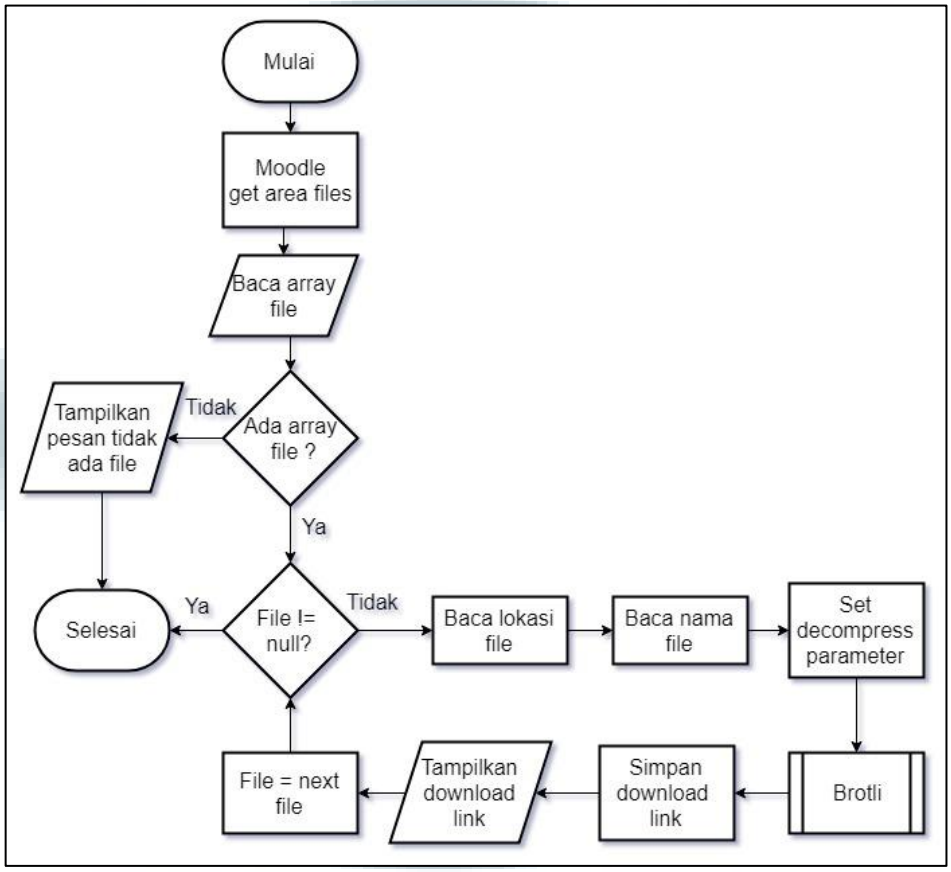


Gambar 3.3 *Flowchart* aplikasi

Pada saat pertama kali dibuka *plugin* akan menampilkan halaman *view*. Disetiap halaman terdapat dua tombol navigasi untuk berpindah, jika tombol *manage files* ditekan maka halaman akan berpindah ke halaman *upload*, jika tombol *view files* ditekan maka halaman akan kembali ke halaman *view*.

Pada modul atau halaman *view*, *plugin* akan mengecek *file* yang tersimpan dan membaca *file* satu-persatu. Setelah membaca lokasi dan nama *file*, *plugin* akan mengatur parameter yang dibutuhkan untuk melakukan dekompresi seperti lokasi *file* dan mode yang akan digunakan. Jika proses dekompresi sudah selesai maka *plugin* akan menampilkan tautan untuk mengunduh *file* tersebut. Proses ini

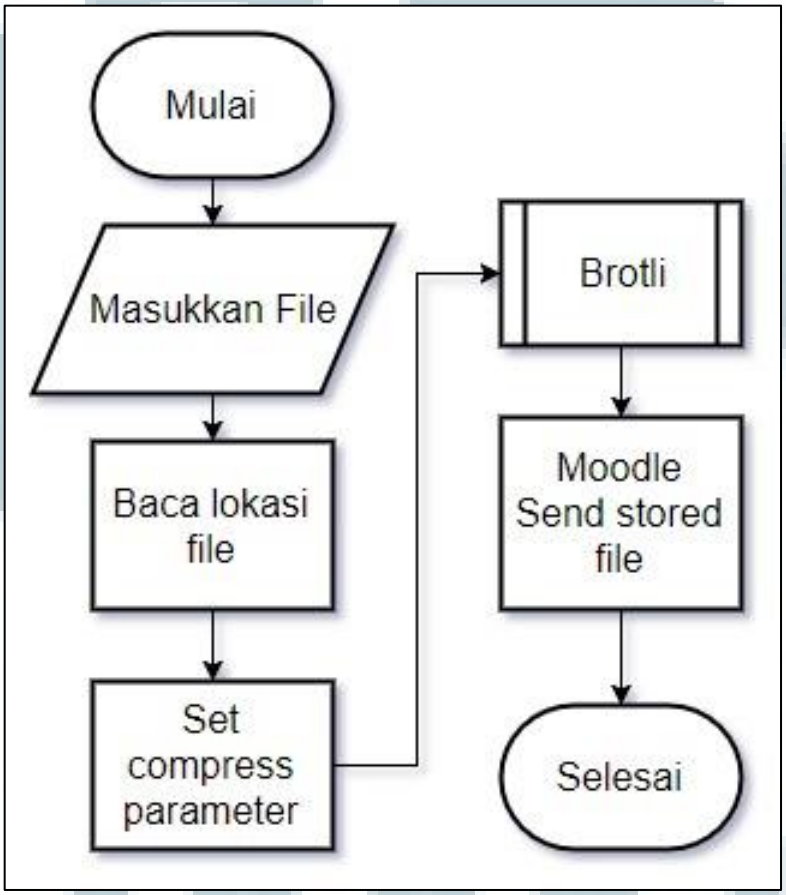
dilakukan pada semua *file* yang tersimpan di dalam *plugin*. Jika tidak ada *file* yang tersimpan, maka *plugin* akan menampilkan pesan bahwa tidak ada *file* yang tersimpan. Untuk lebih jelasnya proses *view* digambarkan pada Gambar 3.4.



Gambar 3.4 *Flowchart view*

Pada modul atau halaman *upload, plugin* menampilkan sebuah halaman dimana pengguna dapat mengunggah *file*. Elemen yang digunakan untuk mengunggah *file* merupakan form *upload* yang dimiliki oleh Moodle, dimana terdapat dua cara untuk melakukan proses unggah. Pertama pengguna dapat mengklik *form* dan memilih *file* dengam menggunakan *file picker* milik Moodle. Kedua pengguna dapat melakukan *drag and drop file* ke *form*. *File* yang sudah diunggah akan muncul di dalam *form*. Pada proses ini *file* sudah dipindahkan sementara ke *direktori sementara* yang terdapat di dalam *plugin* dan saat pengguna

klik tombol *submit plugin* akan membaca lokasi *file* dan nama *file* yang sekarang berada di dalam direktori sementara. Setelah itu *plugin* akan mengatur parameter yang dibutuhkan untuk proses kompresi dan mengirimnya ke dalam proses Brotli. Setelah selesai *file* akan dikirim ke dalam direktori data Moodle oleh proses *Send Stored file*. Proses *upload* digambarkan pada Gambar 3.5.

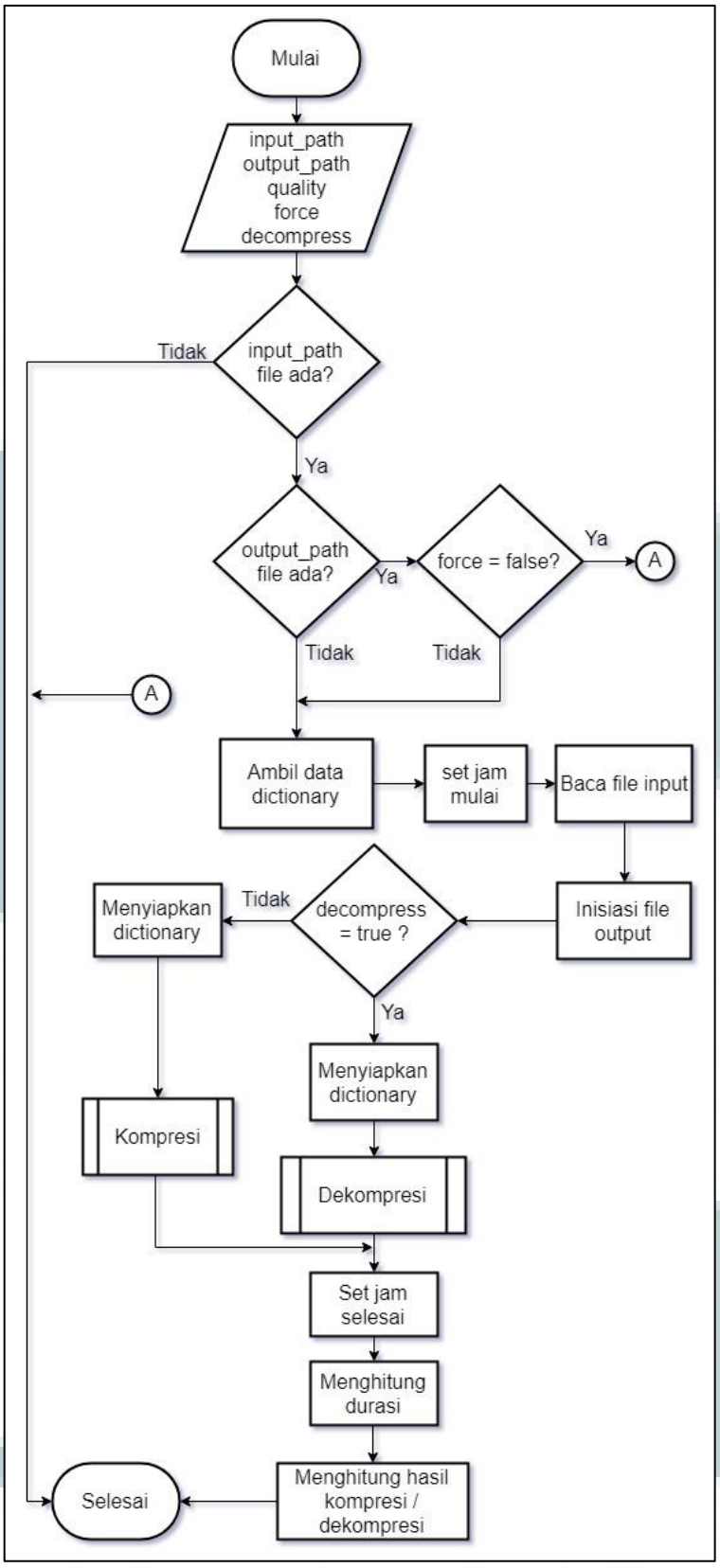


Gambar 3.5 *Flowchart upload*

Pada proses Brotli, *plugin* melakukan pengecekan parameter yang diterima dan menentukan *file* akan dikompresi atau didekompresi. Proses menerima parameter yang dikirim. Proses Brotli akan memvalidasi atau mengecek parameter yang diterima. Jika input_path tidak ada maka proses akan mengembalikan nilai *false*. Jika input_path ada maka proses akan mengecek output_path. Jika

output_path tidak ada maka proses akan mengambil data *dictionary*. Jika output_path ada maka proses akan mengecek variabel *force*. Jika variabel *force* bernilai *false* maka proses akan mengembalikan nilai *false*. Jika variabel *force* bernilai *true* maka proses akan mengambil data *dictionary*, mengambil dan menyimpan waktu mulai, lalu membuka *file* masukkan dan menginisiasi *file* keluaran.

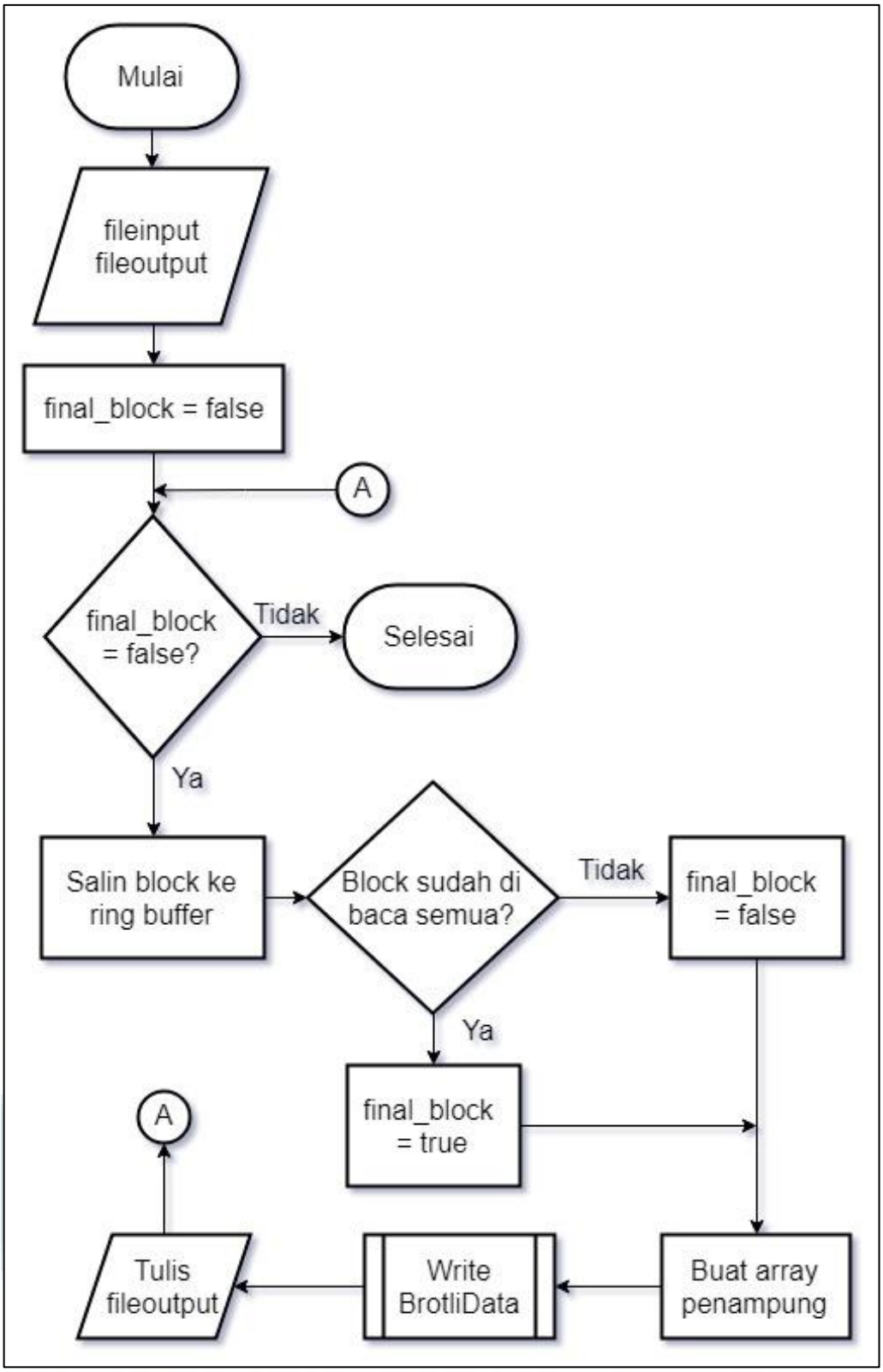
Jika nilai *decompress* sama dengan *true* maka proses mengatur *dictionary* untuk dekompresi dan memanggil proses dekompresi. Jika nilai *decompress* sama dengan *false* maka proses membuka *file dictionary* yang digunakan, mengambil jam mulai, dan memanggil proses kompresi. Setelah proses dekompresi atau kompresi selesai, proses akan mengambil jam selesai, menghitung durasi dengan membandingkannya dengan jam mulai dan menghitung hasil ukuran dari proses kompresi atau dekompresi. Proses Brotli digambarkan pada Gambar 3.6.



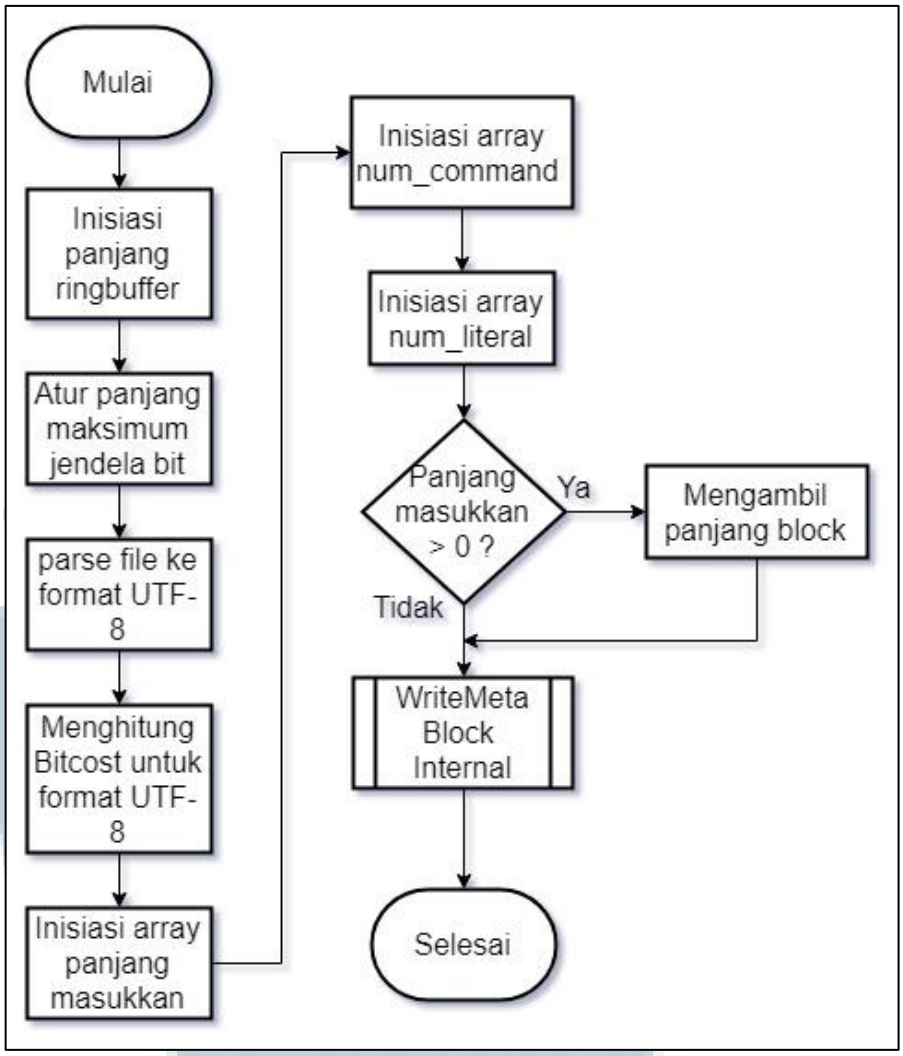
Gambar 3.6 *Flowchart* Brotli

Pada proses kompresi, *plugin* melakukan proses kompresi terhadap *file* masukkan, proses kompresi dilakukan pada setiap *block file*, dari awal *block* hingga akhir *block file*. Proses menerima masukkan *fileinput*, *fileoutput* dan parameter kompresi. Variabel final_block digunakan sebagai *flag* untuk menentukan *block* terakhir pada *file*. Jika final_block sama dengan *false* maka program selesai. Jika tidak, proses memanggil fungsi untuk menyalin *block* pada *file* ke dalam *ring buffer*. Jika jumlah *byte* pada *block* tidak sama dengan 0 dan *file* masukkan sama dengan *null* maka set final_block menjadi *true* jika tidak set final_block menjadi *false* dan jika jumlah *byte* sama dengan 0 maka set final_block menjadi *true*.

Lalu buat *array* untuk menampung data hasil kompresi dan kirim *ring buffer* ke proses WriteBrotlidata. Setelah proses sudah selesai cek panjang dari *array* penampung yang dibuat. Jika lebih besar dari 0 maka panggil fungsi untuk menulis *file* keluaran lalu ulangi *loop*, jika tidak maka ulangi *loop* sampai semua *block* pada *file* terkompresi. Proses kompresi digambarkan pada Gambar 3.7.



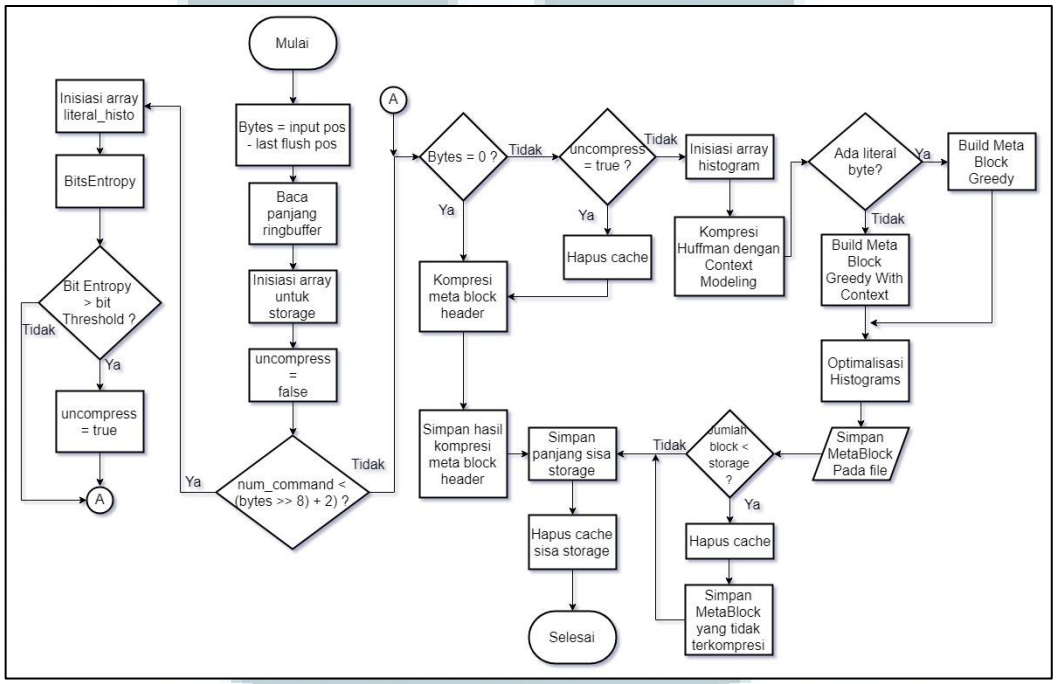
Gambar 3.7 *Flowchart* kompresi



Gambar 3.8 *Flowchart* WriteBrotliData

Gambar 3.8 menjelaskan proses WriteBrotliData. Pada proses ini *plugin* melakukan proses kompresi menggunakan algoritma Brotli. Pertama, aplikasi menginisiasi panjang *ring buffer* baru yang dibutuhkan selama proses kompresi, menghitung dan mengeset panjang maksimum jendela *bit* yang akan digunakan. Panjang maksimum jendela *bit* didapat dari menghitung operasi *bitwise shift* pada variabel lgblock, variabel lgblock adalah panjang dari *block* data, lalu mengubah *block file* yang diterima menjadi format UTF-8, menghitung *bitcost* pada *file*, menginisiasi *array* untuk menampung *block file*, menampung *commands* pada

setiap *block*, dan menampung *bytes* yang tidak bisa dikompresi dengan *backrefering*. Jika panjang *block* masukkan lebih besar dari 0, maka jalankan fungsi GetLengthCode untuk mengambil panjang kode *block* dan jalankan proses WriteMetaBlockInternal. Jika tidak, maka langsung menjalankan proses WriteMetaBlockInternal.



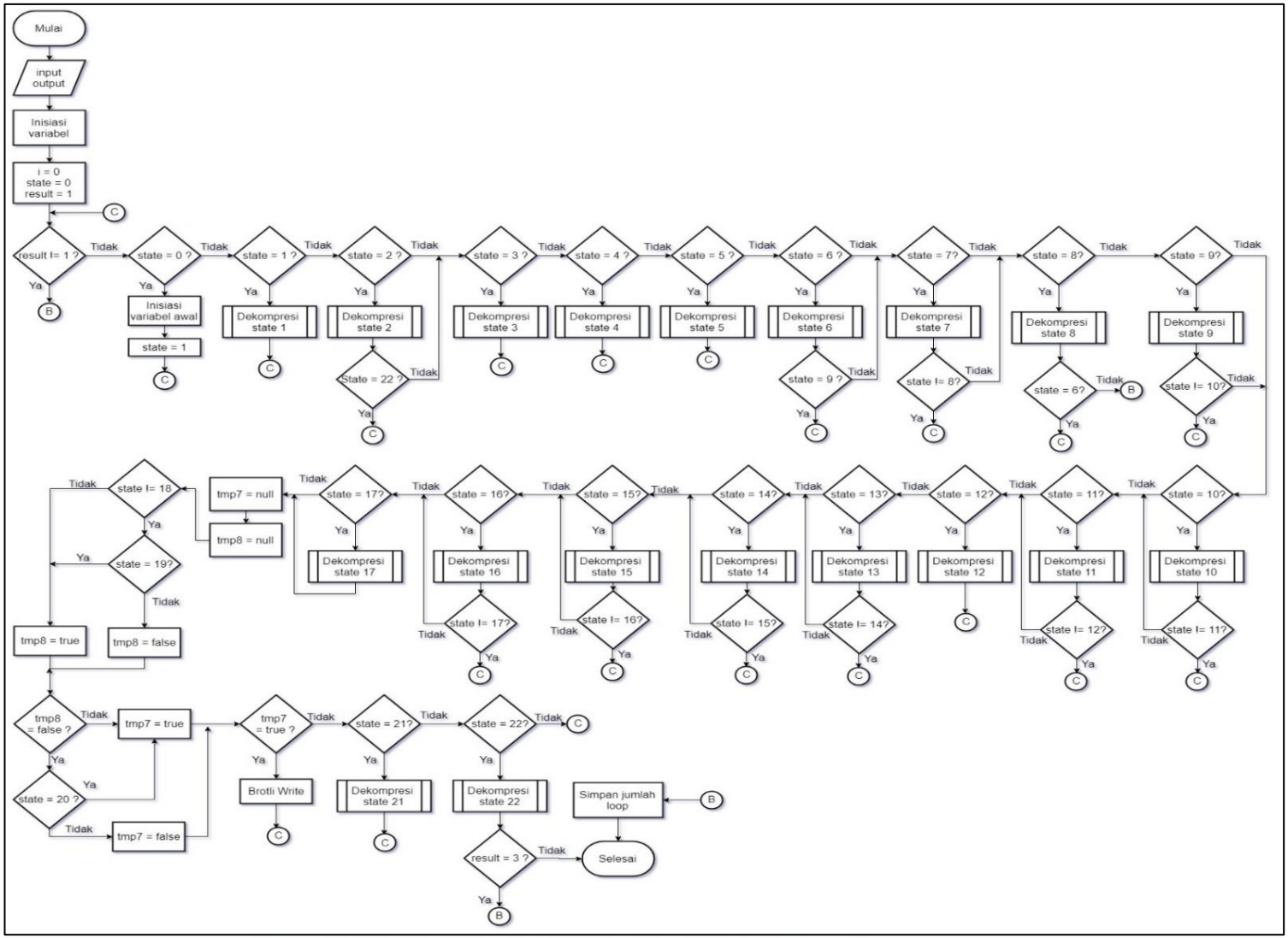
Gambar 3.9 *Flowchart* WriteMetaBlockInternal

Gambar 3.9 menjelaskan proses WriteMetaBlockInternal, proses ini berfungsi untuk mengompresi *block* data dan menuliskannya ke *file* keluaran yang sudah dibuat sebelumnya. Pertama proses akan menghitung *byte* dari posisi masukkan dan posisi terakhir saat melakukan *flush*, membaca panjang dari *ring buffer*, menginisiasi *array* untuk menyimpan *block file*, mengeset variabel *uncompress* menjadi *false*. Variabel *uncompress* digunakan sebagai *flag* yang menandakan kondisi *block* belum terkompresi atau sudah terkompresi. Jika jumlah *command* pada *block* lebih kecil dari nilai (*bytes* >> 8) + 2, maka lakukan

penginisiasian *array* literal_histo untuk menyimpan catatan setiap *byte* yang tidak bisa dikompresi dan memanggil fungsi BitEntropy untuk mengambil *byte* yang tidak dapat dikompresi dari *block* sesuai dengan yang terdapat pada *array* literal_histo. Jika hasil BitEntropy sudah mencapai besar dari jumlah *byte* pada *block file* maka set variabel *uncompress* menjadi *true* dan lanjutkan proses.

Lakukan pengecekan terhadap nilai variabel *bytes*. Jika sama dengan 0, maka lakukan kompresi meta *block header* dan simpan hasil kompresi meta *block header*, simpan panjang sisa *storage* dan hapus *cache* sisa kompresi. Jika variabel *uncompress* sama dengan *true*, maka Hapus *cache*, kompresi meta *block header* dan simpan hasil kompresi meta *block header*, simpan panjang sisa *storage* dan hapus *cache* sisa kompresi. Jika tidak, maka inisiasi *array histogram* untuk merepresentasikan pohon Huffman, jalankan fungsi untuk melakukan kompresi Huffman *code* dengan menggunakan *context modeling*. Fungsi ini melakukan kompresi terhadap setiap huruf atau angka pada *block* dengan menggunakan satu atau lebih pohon Huffman.

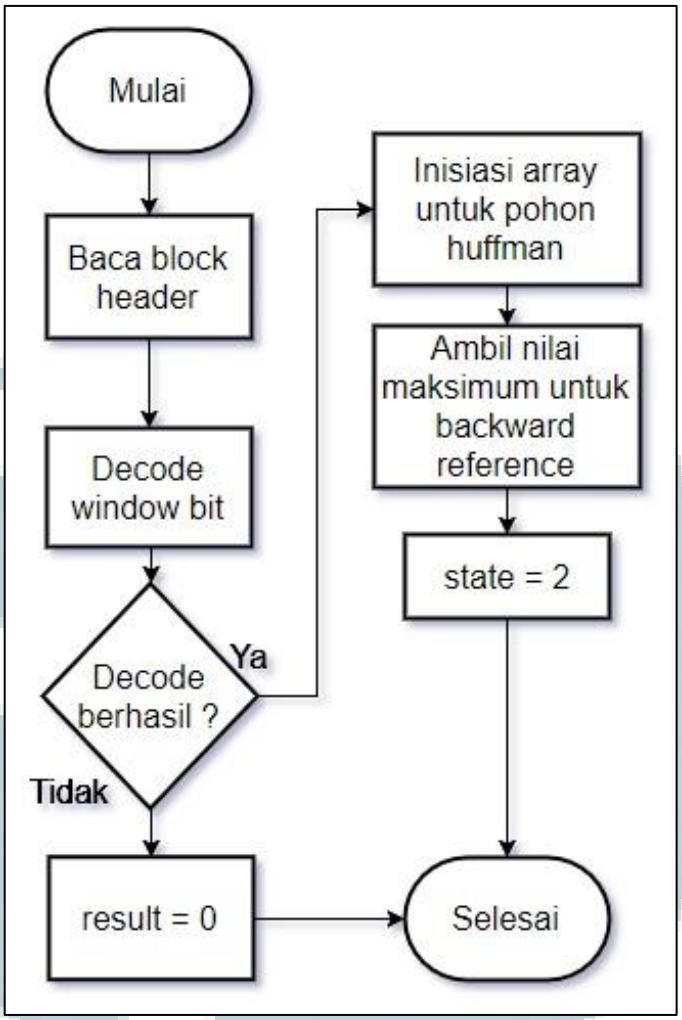
Jika tidak ada *byte* yang tidak dapat dikompresi, maka buat meta *block* yang sudah terkompresi dengan teknik *greedy*. Jika ada *byte* yang tidak bisa di kompresi, maka buat meta *block* yang sudah terkompresi dengan teknik *greedy* dari satu atau lebih pohon Huffman yang digunakan. Lalu lakukan optimalisasi *histogram* dan simpan hasil meta *block* yang sudah terkompresi pada *file*. Jika jumlah *block* lebih kecil dari panjang *array storage*, maka hapus *cache* dan simpan meta *block* yang tidak terkompresi. Lalu simpan panjang sisa *storage* dan hapus *cache* sisa kompresi.



Gambar 3.10 *Flowchart* dekompresi

Gambar 3.10 menggambarkan proses dekompresi dengan algoritma Brotli. Pada proses dekompresi, proses melakukan dekompresi pada *file* yang ingin diunduh. Proses dekompresi Brotli terbagi ke dalam 22 *state*. Setiap *state* digunakan untuk memastikan tidak ada proses *decode* yang salah. Ketika hasil *decode* membutuhkan lebih banyak masukkan, maka setidaknya proses menambahkan satu *byte* untuk menyelesaikan proses *decoding*. Ketika proses dekompresi berhasil, *decoder* harus mengembalikan semua data yang tidak digunakan kembali ke *input buffer*.

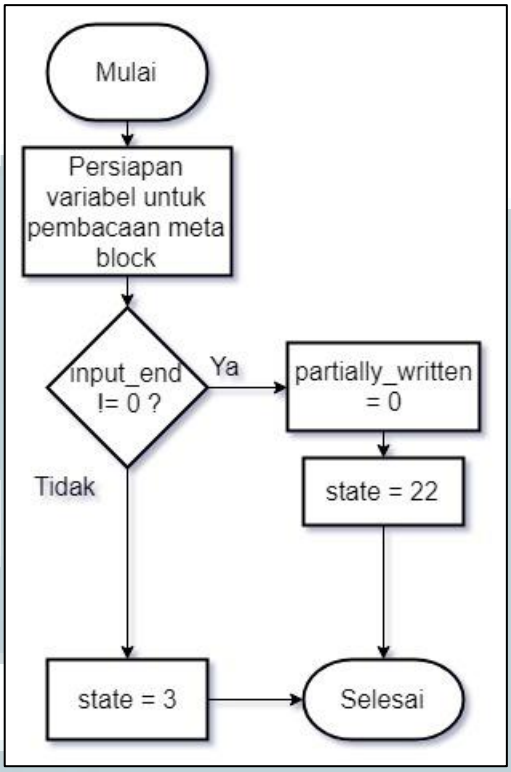
Pertama proses menerima *file* masukkan dan *file* keluaran, menginisiasi variabel yang dibutuhkan dalam proses dekompresi. Variabel *state* digunakan sebagai *flag* yang menandakan langkah dekompresi. Variabel *result* digunakan untuk menandakan apakah *file* sudah berhasil didekompresi atau belum. Jika variabel *result* bernilai 1, maka *file* sudah berhasil didekompresi. Jika variabel *result* bernilai 2, maka proses dekompresi masih memerlukan tambahan masukkan *byte*. Jika variabel result bernilai 3, maka *decoder* mengalami kegagalan saat melakukan pembacaan *file*. Jika variabel *result* bernilai 0, maka proses dekompresi mengalami kegagalan. Proses dekompresi akan selesai sampai seluruh *block* pada *file* sudah didekompresi dan ditulis ke dalam *file* keluaran.



Gambar 3.11 *Flowchart* dekompresi *state* 1

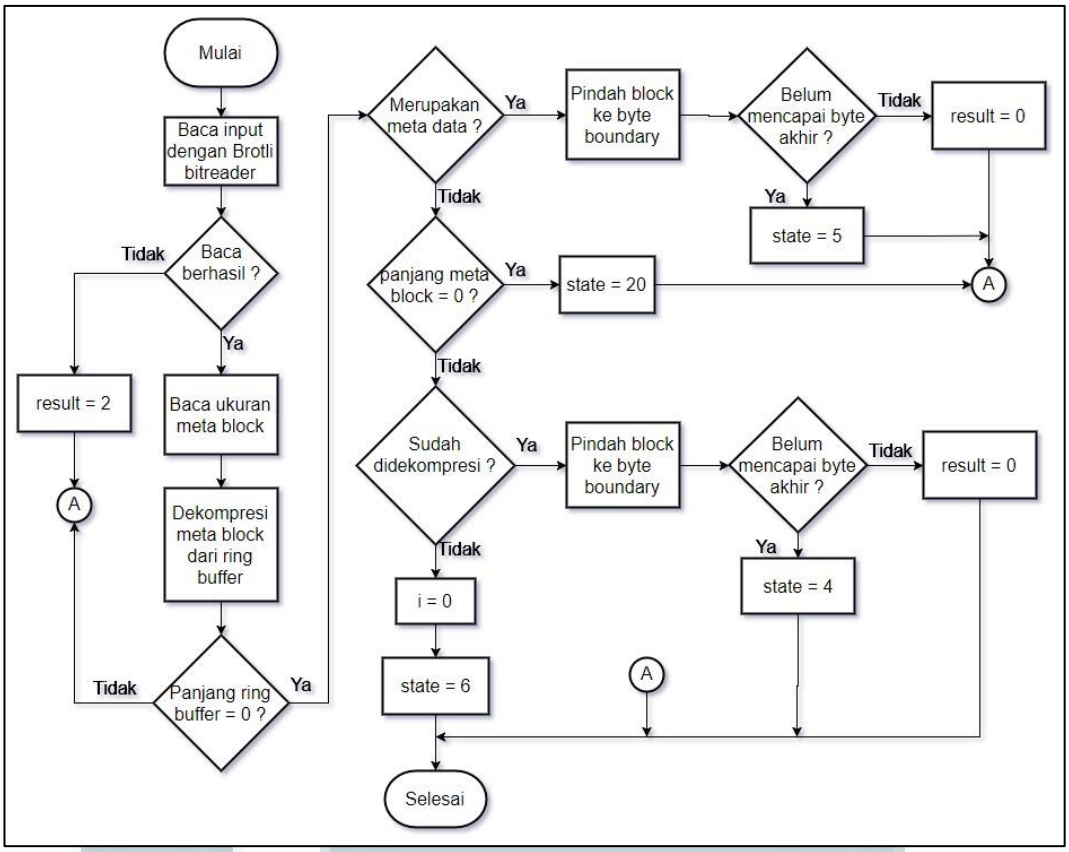
Pada proses dekompresi *state* 1 yang digambarkan pada Gambar 3.11, proses melakukan *decoding* terhadap *header block file*. Proses juga menghitung panjang jendela *bit* maksimum yang digunakan untuk melakukan dekompresi. Panjang jendela *bit* yang dimaksud adalah panjang dari *array* yang digunakan untuk merepresentasikan *sliding window*. Pertama proses membaca *header block* dengan Brotli Bitreader, Brotli BitReader berfungsi untuk membaca *file* masukkan dalam bentuk *bit*, ukuran yang dibaca bervariasi tergantung *block* yang ingin dibaca. Jika proses membaca *bit* gagal, maka set variabel *result* menjadi 2. Jika berhasil membaca, maka panggil fungsi untuk melakukan *decoding* terhadap *header block*

file. Jika *decode* berhasil, maka set variabel *result* menjadi 0. Jika tidak, maka inisiasi *array* untuk pohon Huffman, dan set variabel *state* menjadi 2.



Gambar 3.12 *Flowchart* dekompresi *state* 2

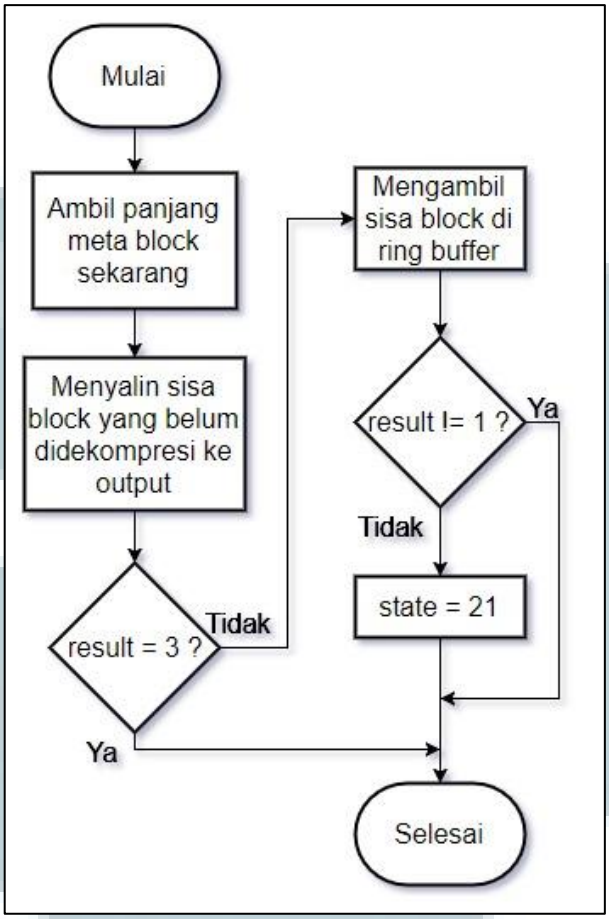
Pada proses dekompresi *state* 2 yang digambarkan pada Gambar 3.12, proses akan mengecek dan menentukan apakah *file* sudah selesai didekompresi atau belum. Variabel input_end berfungsi sebagai *flag* yang menandakan kondisi dimana pembacaan *file* sudah selesai atau belum selesai. Jika pembacaan sudah selesai maka fungsi akan mengeset variabel *state* menjadi 22. Jika belum selesai, maka fungsi akan mengeset variabel *state* menjadi 3.



Gambar 3.13 *Flowchart* dekompresi *state* 3

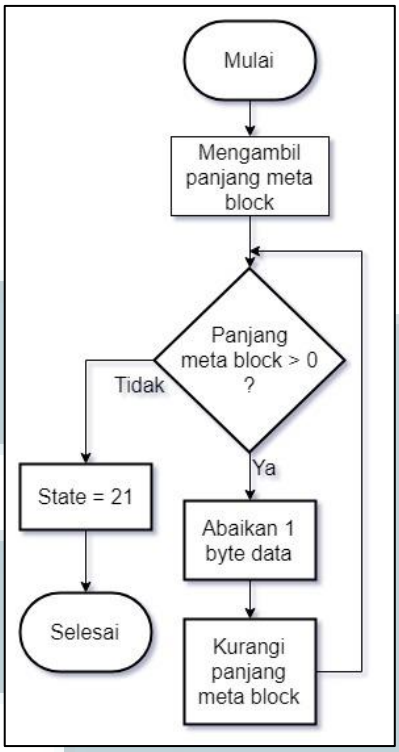
Pada proses dekompresi *state* 3 yang digambarkan pada Gambar 3.13, proses melakukan dekompresi *meta-block* pada *file* masukkan. Pertama proses membaca *file* masukkan dengan Brotli Bitreader. Jika gagal, maka set variabel *result* menjadi 2. Jika berhasil maka baca ukuran meta *block* dan lakukan dekompresi meta *block* dari *ring buffer*. Jika panjang *ring buffer* belum kosong, maka set variabel *result* menjadi 2. Jika *block* yang didekompresi merupakan *meta* data, maka panggil fungsi untuk mengganti *block file*. Jika belum mencapai *block* akhir, maka set variabel *state* menjadi 5. Jika tidak ditemukan set variabel *result* menjadi 0. Jika panjang *meta-block* sama dengan 0, maka set variabel *state* menjadi 21. Jika meta *block* sudah didekompresi, maka panggil fungsi untuk mengganti *block file*. Jika belum mencapai *block* akhir, maka set variabel *result* menjadi 0. Jika

ditemukan set variabel *state* menjadi 4. Jika meta *block* tidak terdekompresi, maka set i menjadi 0, dan set variabel *state* menjadi 6.



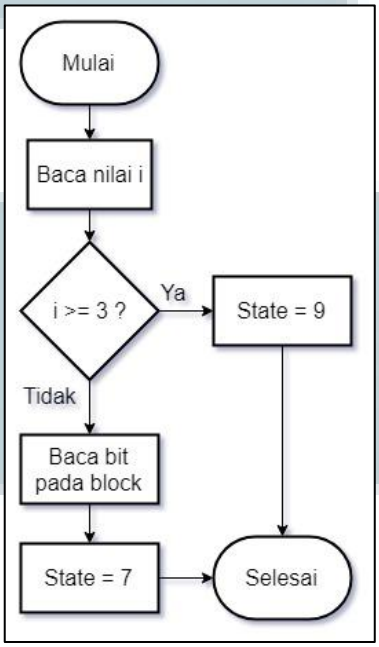
Gambar 3.14 *Flowchart* dekompresi *state* 4

Pada proses dekompresi *state* 4 yang digambarkan pada Gambar 3.14, Proses berfungsi untuk mengambil sisa *block* yang belum didekompresi dan mengambil sisa *block* pada *ring buffer*. Pertama proses menginisiasi panjang meta *block* sekarang, membaca sisa *block* yang belum didekompresi. Jika *result* sama dengan 3, maka selesai. Jika tidak, maka hapus sisa *block* pada *ring buffer*. Jika *result* tidak sama dengan 1, maka selesai. Jika sama dengan 1, maka set variabel *state* menjadi 21.



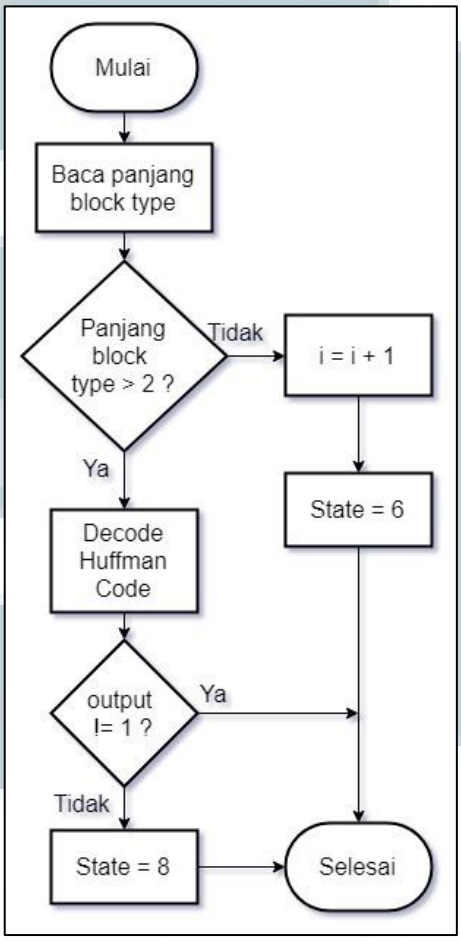
Gambar 3.15 *Flowchart* dekompresi *state* 5

Pada proses dekompresi *state* 5 yang digambarkan pada Gambar 3.15, proses membaca setiap *meta-block* dan mengabaikannya. Proses akan terus mengulang sampai panjang *meta-block* sama dengan 0.



Gambar 3.16 *Flowchart* dekompresi *state* 6

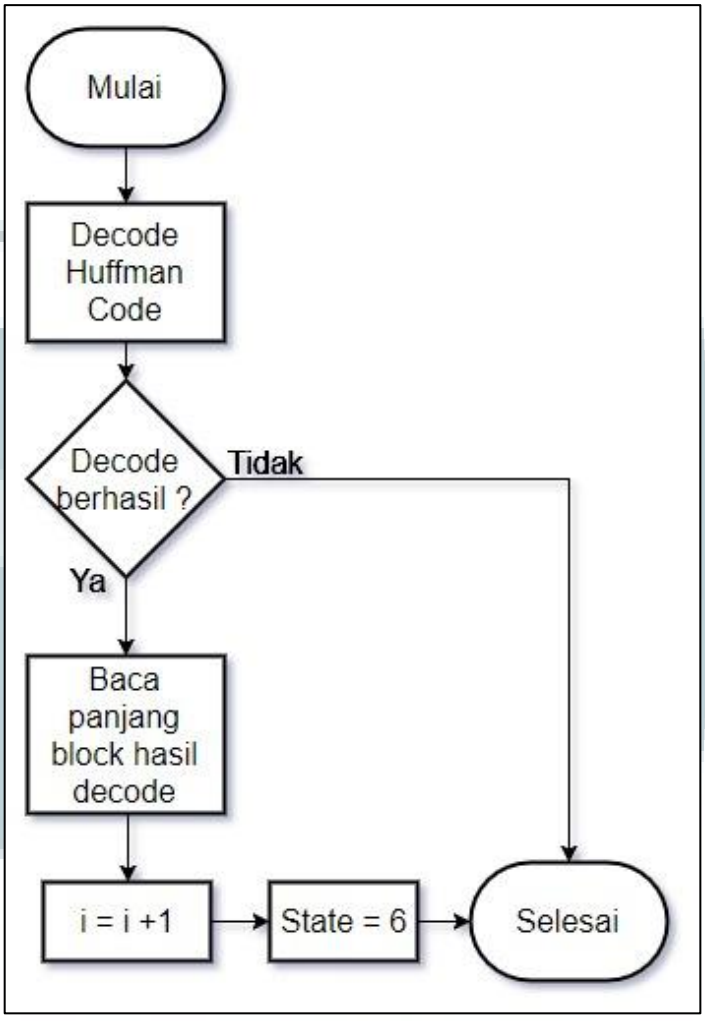
Pada proses dekompresi *state* 6 yang digambarkan pada Gambar 3.16, proses menyimpan panjang *block bit* sekarang. Pertama proses mengecek nilai dari i. Jika i lebih besar atau sama dengan 3, maka set variabel *state* menjadi 9. Jika tidak, maka simpan panjang bit sekarang dan set variabel *state* menjadi 7.



Gambar 3.17 *Flowchart* dekompresi *state* 7

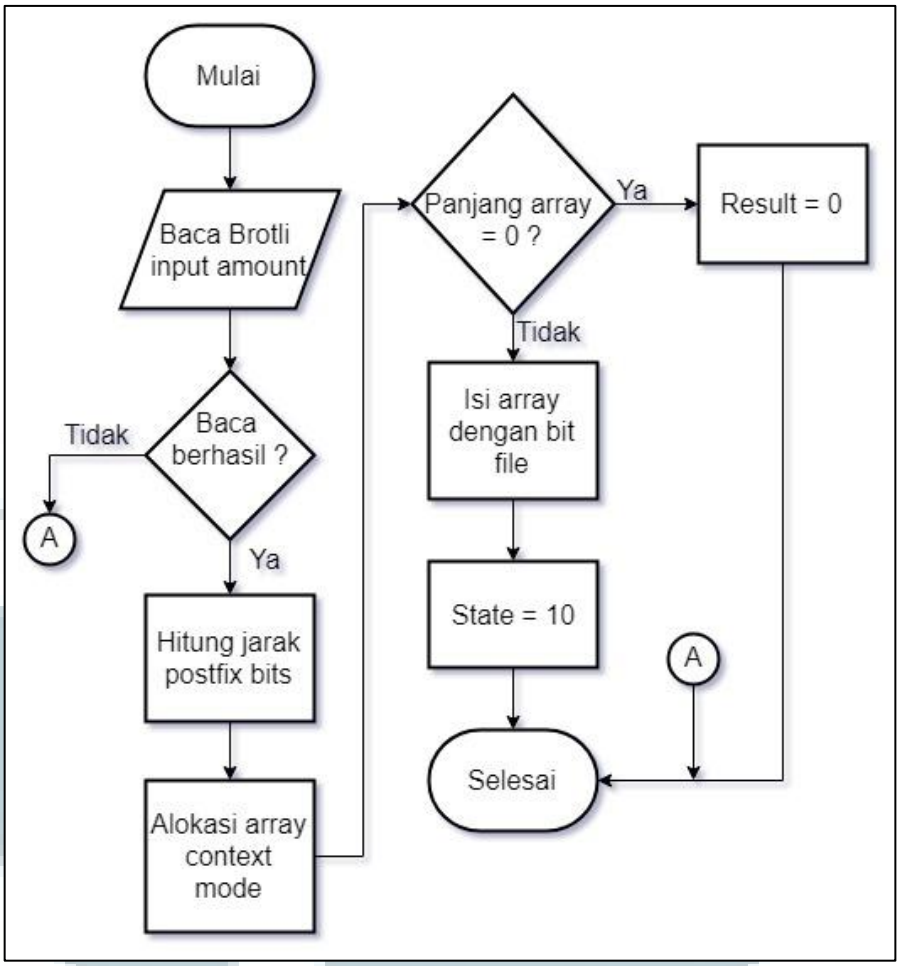
Pada proses dekompresi *state* 7 yang digambarkan pada Gambar 3.17, proses melakukan dekompresi menggunakan pohon Huffman. Pertama proses mengecek panjang *block type*, *block type* adalah tipe *block* yang tersimpan pada *file* saat proses kompresi berlangsung. Jika panjangnya tidak lebih besar dari 2, maka tambahkan nilai i, set variabel *state* menjadi 6. Jika panjang lebih besar dari 2, maka *decode block* huffman *code* dengan memanggil fungsi untuk melakukan proses

decode Huffman *code*. Jika *output* sama dengan 1, maka set variabel *state* menjadi 8. Jika tidak sama dengan 1, maka selesai.



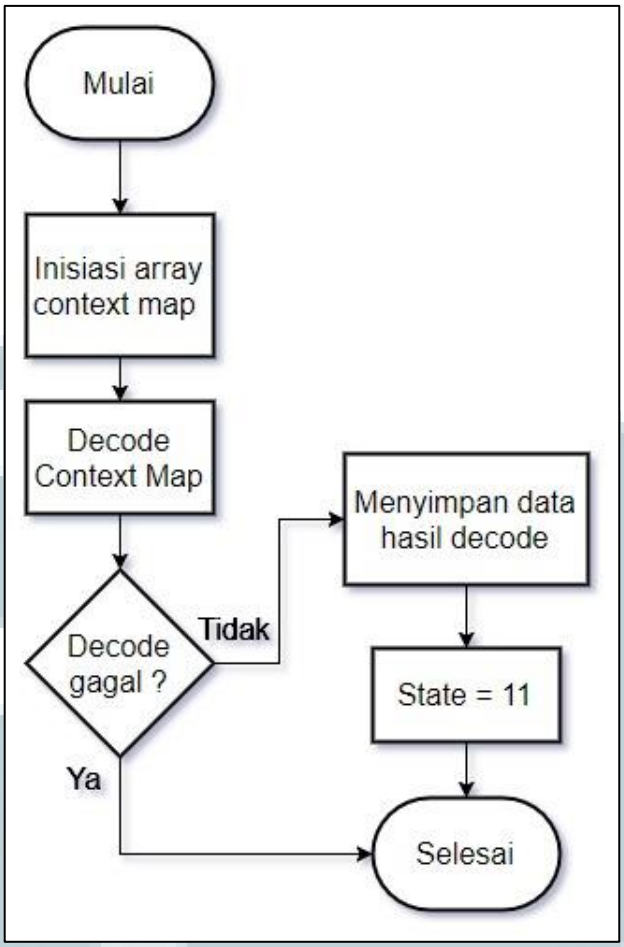
Gambar 3.18 *Flowchart* dekompresi *state* 8

Pada proses dekompresi *state* 8 yang digambarkan pada Gambar 3.18, proses berfungsi untuk melakukan *decoding* Huffman *code* terhadap *block* data sekarang. Pertama proses melakukan *decode block* dengan Huffman *code*. Jika *decode* gagal, maka selesai. Jika *decode* berhasil, maka baca panjang *block* hasil *decode* dan simpan, tambah nilai i dengan 1, dan set variabel *state* dengan 6.



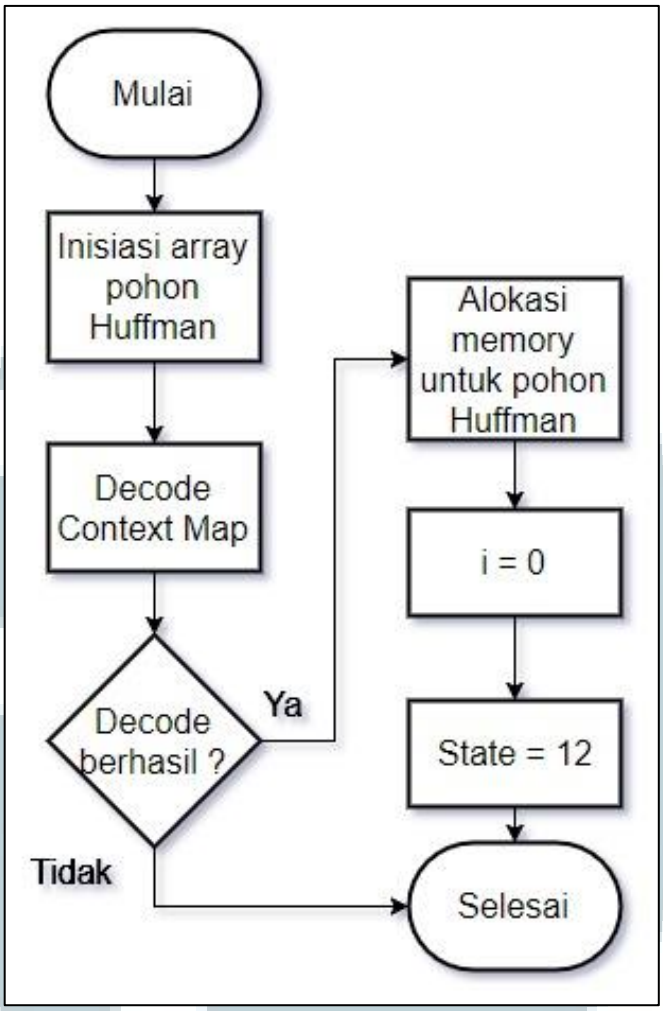
Gambar 3.19 *Flowchart* dekompresi *state* 9

Pada proses dekompresi *state* 9 yang digambarkan pada Gambar 3.19, proses berfungsi untuk membaca *block* yang sudah didekompresi dan menyimpannya ke dalam *array file*. Pertama proses membaca sisa *block* dengan memanggil fungsi *BrotliInputAmount*. Jika baca gagal, maka selesai. Jika berhasil, maka hitung jarak *postfix* bit pada *block* dan alokasi *array* untuk *context mode*. Jika panjang *array* sama dengan 0, maka set *result* menjadi 0 dan selesai. Jika tidak, maka isi *array* dengan *bit tree*, dan set variabel *state* menjadi 10.



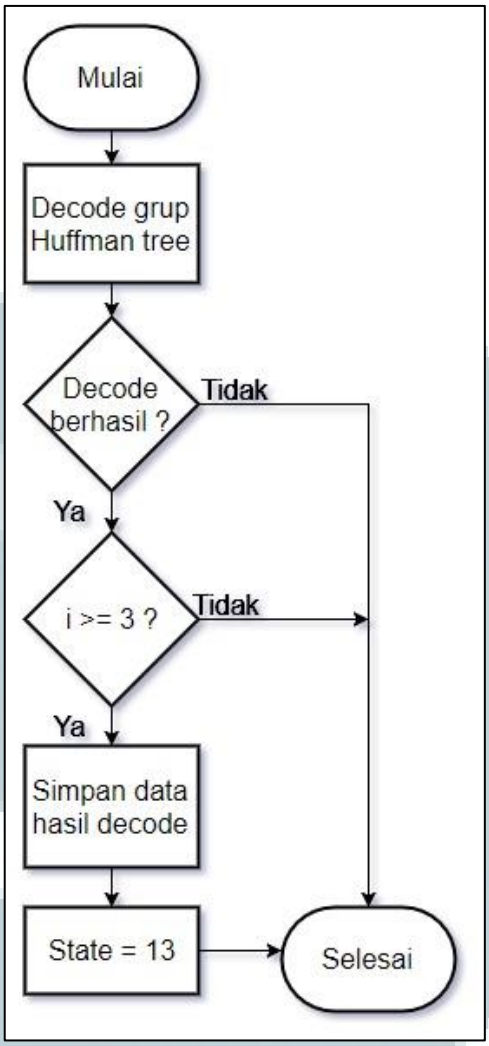
Gambar 3.20 *Flowchart* dekompresi *state* 10

Pada proses dekompresi *state* 10 yang digambarkan pada Gambar 3.20, proses berfungsi untuk melakukan *decode block* simbol menggunakan Huffman *code* dengan *context modeling*. Pertama proses menginisiasi *array* untuk *context map* dan jalankan fungsi *decode context map*, fungsi ini berfungsi untuk melakukan *decoding* menggunakan satu atau lebih pohon Huffman untuk mendapatkan simbol pada *block* yang terkompresi. Jika *decode* gagal, maka selesai. Jika *decode* berhasil, maka simpan data hasil *decode* dan set variabel *state* menjadi 11.



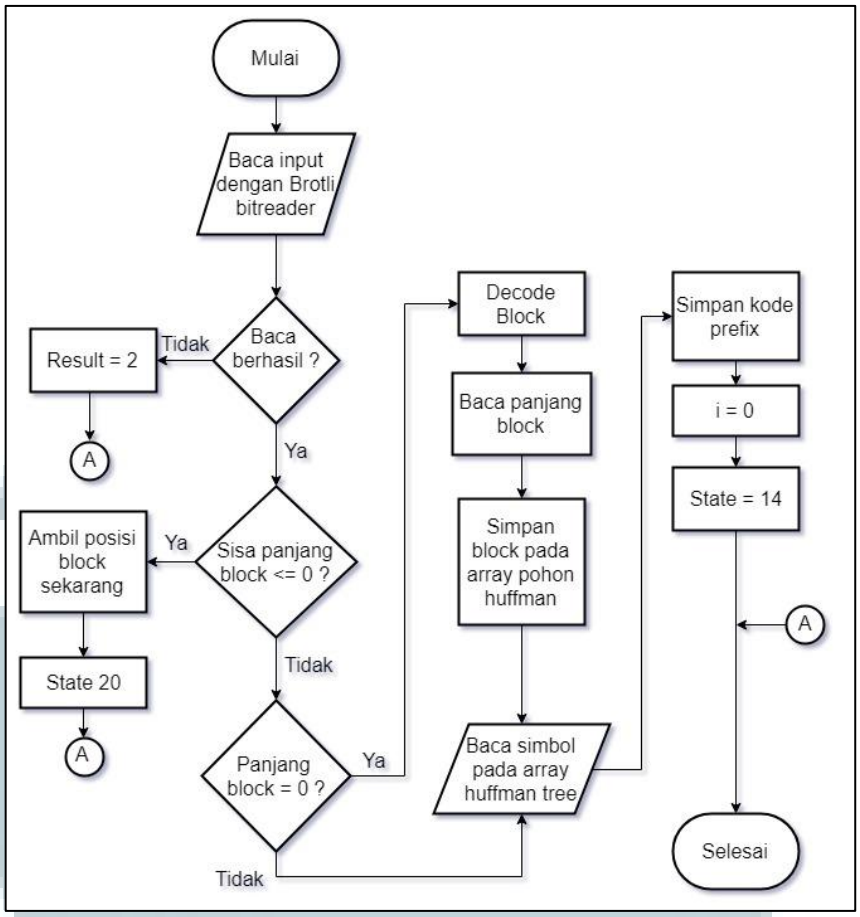
Gambar 3.21 *Flowchart* dekompresi *state* 11

Pada proses dekompresi *state* 11 yang digambarkan pada Gambar 3.21, proses berfungsi untuk melakukan *decoding* menggunakan Huffman *code* dengan distribusi *context map*. Pertama proses menginisiasi *array* untuk merepresentasikan pohon Huffman dan panggil fungsi *decode context map*, Jika *decode* gagal, maka selesai. Jika *decode* berhasil, maka inisiasi grup untuk pohon Huffman, set i menjadi 0, dan set variabel *state* menjadi 12.



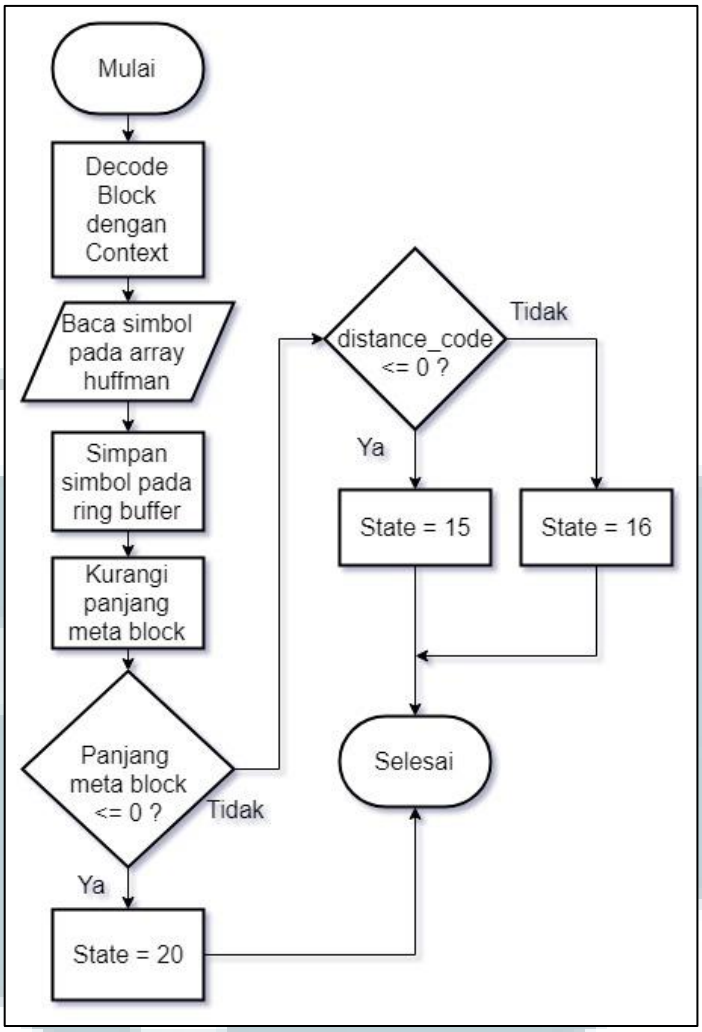
Gambar 3.22 *Flowchart* dekompresi *state* 12

Pada proses dekompresi *state* 12 yang digambarkan pada Gambar 3.22, proses berfungsi untuk melakukan *decoding* Huffman *code* pada *block file*. Pertama proses memanggil fungsi untuk melakukan *decoding* terhadap *block file*. Jika *decode* gagal, maka selesai. Jika i tidak lebih besar atau sama dengan 3, maka selesai. Jika lebih besar atau sama dengan 3, maka set variabel *state* menjadi 13.



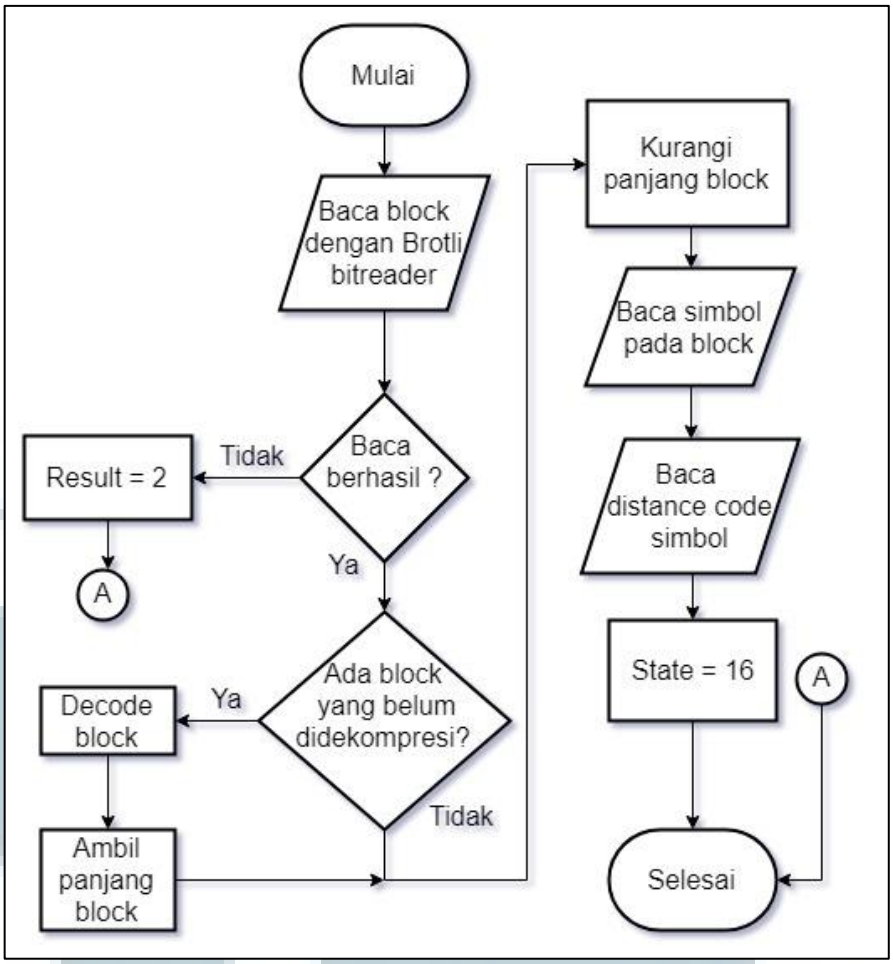
Gambar 3.23 *Flowchart* dekompresi *state* 13

Pada proses dekompresi *state* 13 yang digambarkan pada Gambar 3.23, proses berfungsi untuk melakukan *decoding* terhadap *block type* dan mengambil kode *prefix* dari hasil *decode*. Pertama proses membaca *block file*. Jika baca gagal, maka set variabel *result* menjadi 2 dan selesai. Jika sisa panjang *block* lebih kecil dari 0, maka ambil posisi *block*, set variabel *state* menjadi 20, dan selesai. Jika panjang *block* sama dengan 0, maka *decode block*, baca panjang *block*, dan simpan *block* pada *array* pohon huffman. Baca setiap simbol yang terdapat pada *array* pohon Huffman, simpan kode *prefix*, set i menjadi 0, dan set variabel *state* menjadi 14.



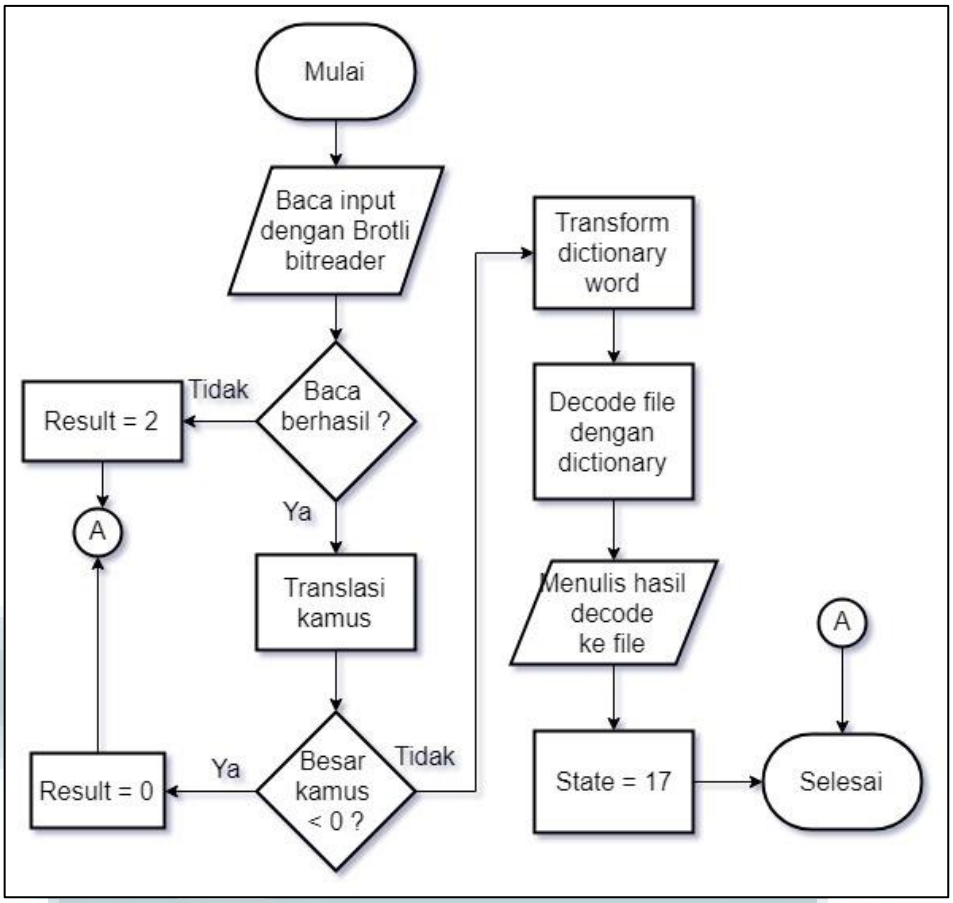
Gambar 3.24 *Flowchart* dekompresi *state* 14

Pada proses dekompresi *state* 14 yang digambarkan pada Gambar 3.24, proses berfungsi untuk melakukan *decoding* terhadap *meta-block* yang terdapat pada *ring buffer*. Pertama proses memanggil fungsi untuk *decode block type context* pada *ring buffer*, baca simbol pada *array* Huffman, simpan simbol pada *ring buffer*, dan kurangi panjang *meta-block*. Jika sisa panjang *meta-block* lebih kecil atau sama dengan 0, maka set variabel *state* menjadi 20 dan selesai. Jika *distance code* lebih besar dari 0, maka set variabel *state* menjadi 16 dan selesai. Jika tidak maka set variabel *state* menjadi 15.



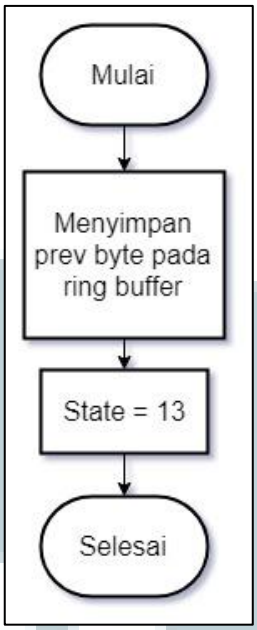
Gambar 3.25 *Flowchart* dekompresi *state* 15

Pada proses dekompresi *state* 15 yang digambarkan pada Gambar 3.25, proses berfungsi untuk membaca *code distance* simbol yang terdapat pada *block* yang sudah *decode*. Pertama proses membaca *block* yang sudah *decode* dengan Brotli BitReader. Jika baca gagal, maka set variabel *result* menjadi 2 dan selesai. Jika ada *block* yang belum didekompresi, maka *decode block*. Kurangi panjang *block* sebesar 1, baca simbol pada *block*, baca *distance code* simbol, dan set variabel *state* menjadi 16.



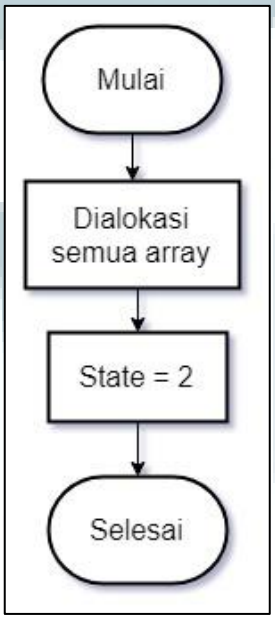
Gambar 3.26 *Flowchart* dekompresi *state* 16

Pada proses dekompresi *state* 16 yang digambarkan pada Gambar 3.26, proses berfungsi untuk melakukan *decode* menggunakan kamus statik milik Brotli. Pertama proses membaca *block*. Jika baca gagal, maka set *result* menjadi 2 dan selesai. Lakukan translasi kamus untuk mentranslasi kamus yang di *hash*. Jika isi kamus lebih kecil dari 0, maka set *result* menjadi 0. Mengubah kamus kata yang ter-*hash* menjadi kata untuk men-*decode block*, lakukan *decode block* dengan data dari kamus, menulis hasil *decode* ke *file* keluaran, dan set variabel *state* menjadi 17.



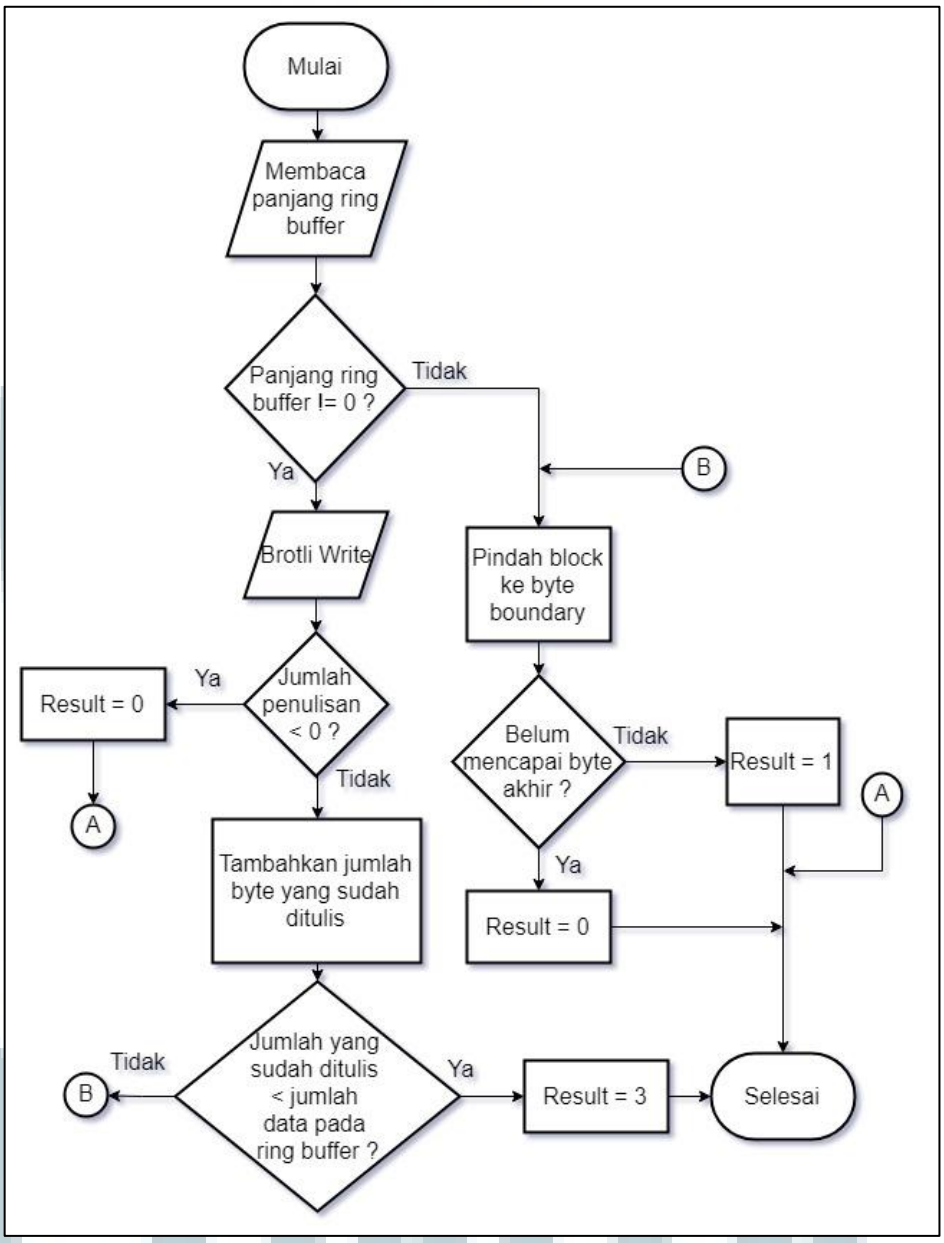
Gambar 3.27 *Flowchart* dekompresi *state* 17

Pada proses dekompresi *state* 17 yang digambarkan pada Gambar 3.27, proses berfungsi untuk menyimpan *byte* terakhir yang dibaca pada *ring buffer*. Proses akan menyimpan *prev byte* pada *ring buffer* dan set variabel *state* menjadi 13.



Gambar 3.28 *Flowchart* dekompresi *state* 21

Pada proses dekompresi *state* 21 yang digambarkan pada Gambar 3.28, proses berfungsi untuk menghapus seluruh *array* yang sudah tidak terpakai. Proses ini dilakukan untuk menyiapkan proses dekompresi pada *block file* berikutnya.



Gambar 3.29 *Flowchart* dekompresi *state* 22

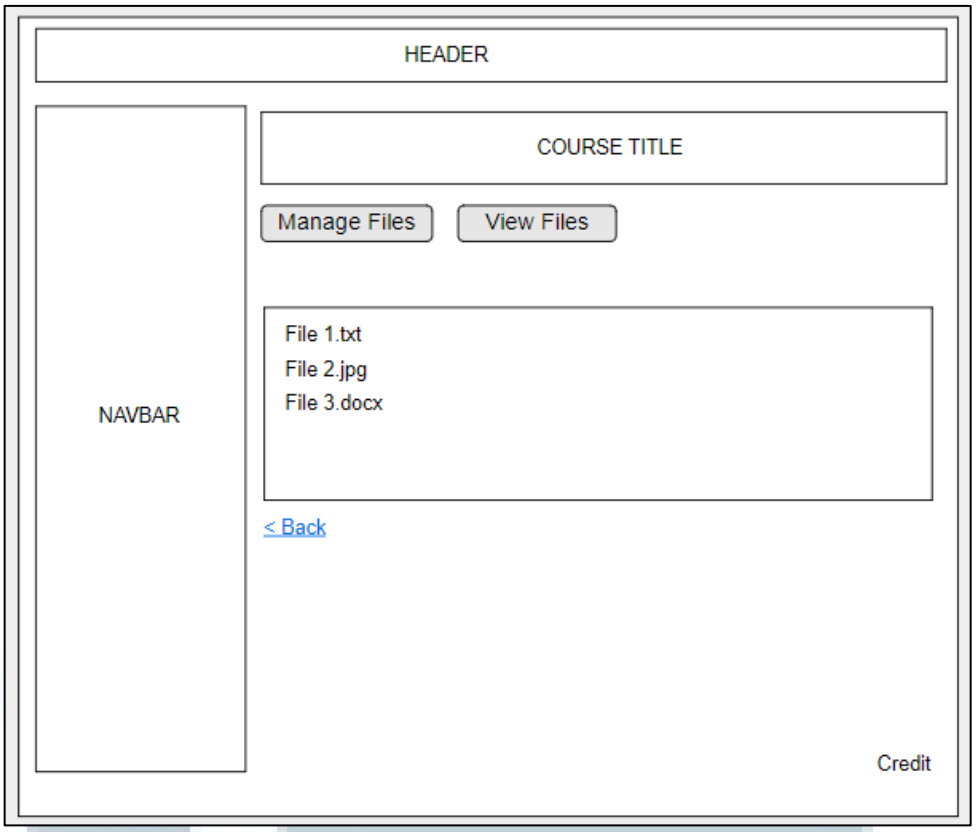
Pada proses dekompresi *state* 22 yang digambarkan pada Gambar 3.29, proses berfungsi untuk menulis *block* yang sudah terdekompresi dari dalam *ring buffer* ke *file* keluaran. Pertama proses melakukan pengecekan kepada panjang *ring*

buffer sekarang. Jika panjang sama dengan 0, maka pindahkan *block* ke *byte boundary*. Jika belum mencapai *byte* akhir, maka set variabel *result* menjadi 0. Jika panjang tidak sama dengan 0, maka panggil fungsi BrotliWrite untuk menulis *block* ke *file* keluaran. Jika jumlah yang ditulis lebih kecil dari 0, maka set variable *result* menjadi 0. Jika tidak, maka tambahkan jumlah *byte* yang sudah ditulis. Jika jumlah yang sudah ditulis lebih kecil dari jumlah data pada *ring buffer*, maka set variabel *result* menjadi 3 dan selesai.

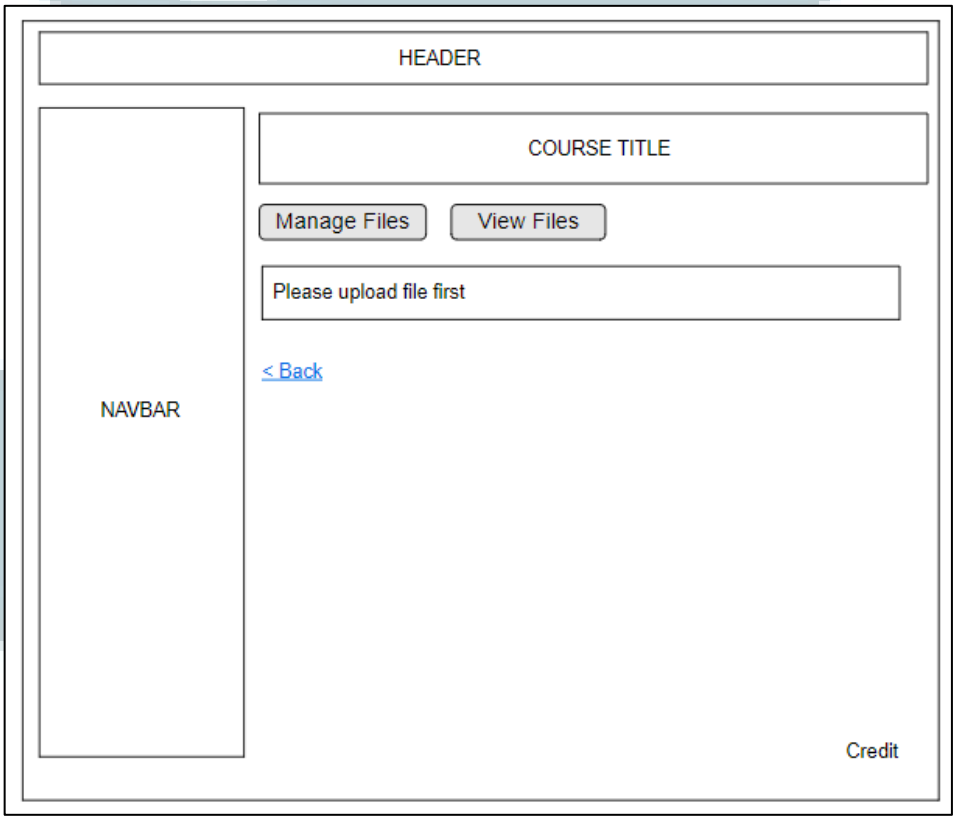
3.3.2 Perancangan Antarmuka

Desain tampilan antarmuka untuk *plugin* ini dibagi menjadi enam yaitu halaman *Upload*, halaman *View*, halaman Unggah Berhasil, halaman Tidak Ada *File*, halaman Menunggu *File*, dan halaman Unggah Dibatalkan. Pada setiap halaman terdapat dua tombol navigasi yaitu tombol *manage file* yang berguna untuk berpindah ke halaman *Upload* dan tombol *view files* yang berguna untuk berpindah ke halaman *View* dan terdapat *credit* di pojok kanan bawah.

Tampilan pertama saat *plugin* dibuka adalah halaman *View* dimana di halaman ini ditampilkan *file* yang sudah diunggah sebelumnya seperti pada Gambar 3.30. Jika belum ada *file* yang diunggah maka halaman akan memunculkan pesan bahwa tidak ada *file* dan menyarankan pengguna untuk mengunggah *file* seperti pada Gambar 3.31.

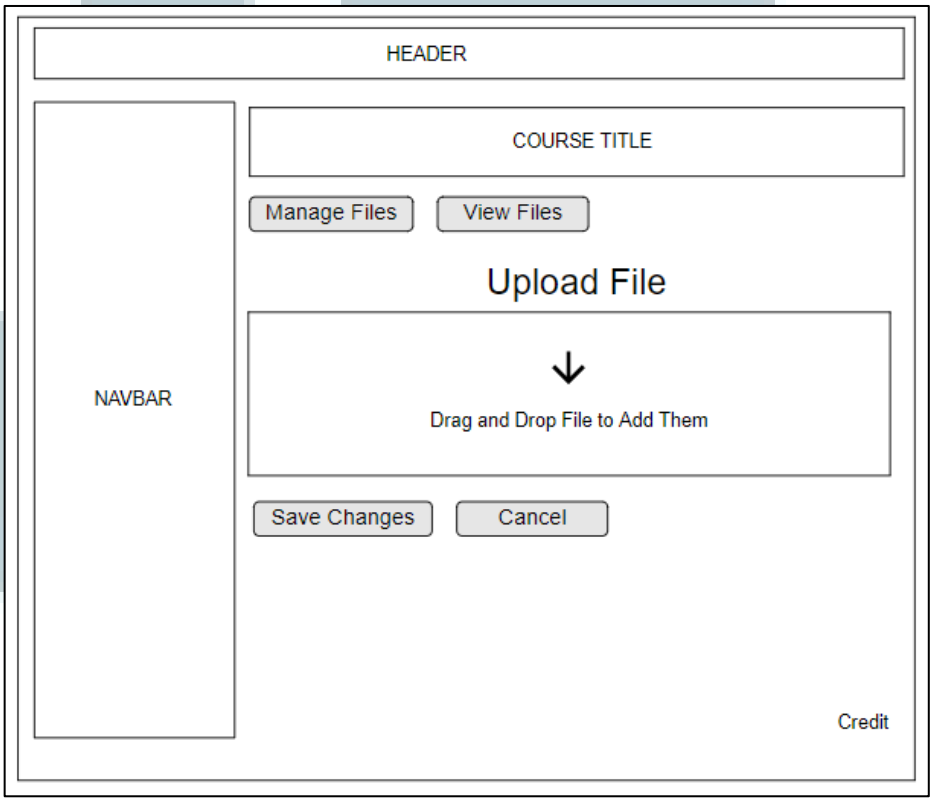


Gambar 3.30 Rancangan Halaman *View*

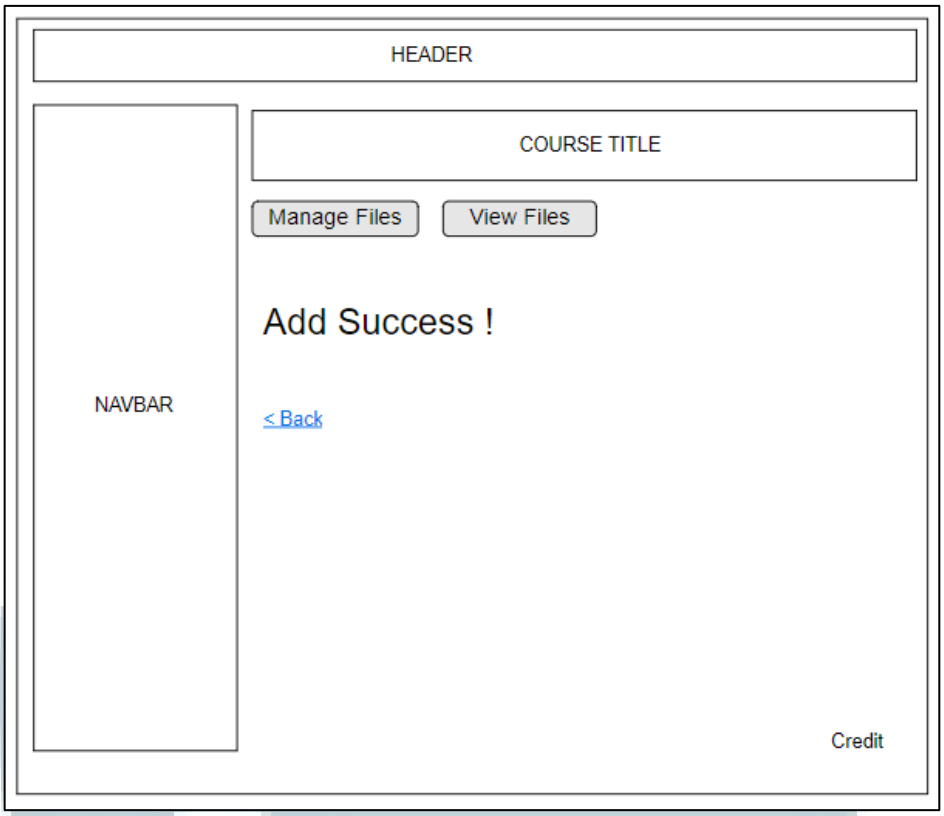


Gambar 3.31 Rancangan Halaman Tidak Ada *File*

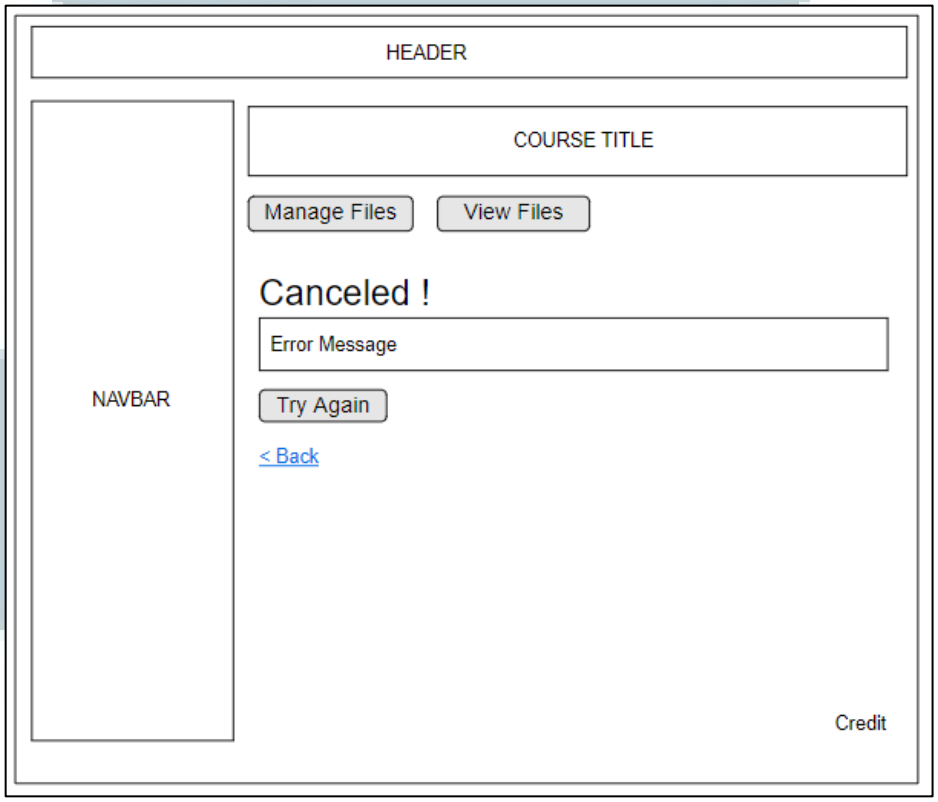
Pada halaman *View*, terdapat sebuah *list item* di tengah halaman, *list* tersebut berisi *file*-file yang sudah diunggah oleh pengguna. Jika *list item* diklik maka akan mengunduh *file* yang dipilih. Jika tidak ada *file* yang diunggah maka akan menampilkan pesan bahwa belum ada *file* dan menyarankan pengguna untuk mengunggah *file* seperti pada Gambar 3.31. Pada halaman *Upload*, terdapat sebuah *form* di tengah yang berfungsi untuk mengunggah *file*. *Form* yang digunakan untuk mengunggah *file* merupakan elemen milik Moodle. Terdapat dua cara untuk mengunggah *file*, pertama dengan mengklik *form* dan memilih *file* dengan *file picker dialog*, kedua dengan melakukan *drag and drop file* yang ingin diunggah ke *form*. Halaman *Upload* digambarkan pada gambar 3.32. Jika pengguna sudah selesai mengunggah *file*, pengguna dapat menekan tombol *save changes* untuk menyimpan *file* ke dalam Moodle ditampilkan seperti Gambar 3.33 dan tombol *cancel* untuk membatalkan aksi seperti pada Gambar 3.34.



Gambar 3.32 Rancangan Halaman *Upload*



Gambar 3.33 Rancangan Halaman Unggah Berhasil



Gambar 3.34 Rancangan Halaman Unggah Dibatalkan