BAB II
TEORI PENUNJANG

2.1 Penelitian Terdahulu

Beberapa penelitian tentang *QR Code* yang telah dilakukan menggunakan kompresi data untuk peningkatan jumlah informasi yang disimpan. Berikut ini merupakan beberapa penelitian yang membahas penerapan kompresi data informasi pada *QR Code*.

Penelitian pertama merupakan thesis dari Wahyu Juliarianto pada tahun 2015 di IPB. Penelitian ini membahas penerapan algoritma kompresi data LZW pada *QR Code*. Percobaan menggunakan identitas data diri (KTP, KK, SIM) yang disimpan dalam bentuk txt. Hasilnya adalah jika ukuran *file* yang akan dimampatkan kecil, maka kompresi tidak berhasil karena ukurannya semakin bertambah, namun jika semakin besar ukuran *file*nya, maka kemungkinan berhasilnya semakin tinggi. [4]

Penelitian kedua berjudul *Increase Capacity of QR Code Using Copression Technique* membahas bagaimana mengkombinasikan kompresi data *lossless* dengan modifikasi *QR Code* dengan warna. Hasilnya adalah terlihat adanya peningkatan daya tampung informasi pada *QR Code* hingga 14% sampai 29%. [5]

Selanjutnya merupakan penelitian dari Nancy Victor dari *Center for Information Technology and Engineering*, M.S University. Penelitian ini membahas bagaimana caranya meningkatkan kapasitas *QR Code* dengan menggunakan kompresi data untuk meningkatkan kapasitas penyimpanan pada *QR Code*. [6]

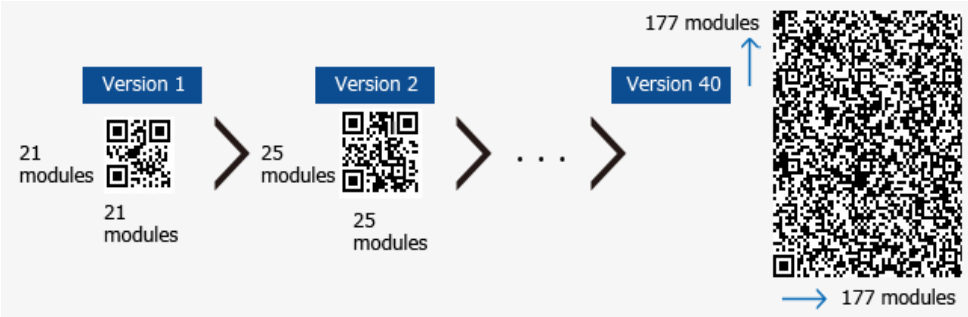
Yang keempat merupakan penelitian yang berjudul *Expanding the Data Capacity of QR Code Using Multiple Compression Algorithms and Base64 Encode/Decode.* Penelitian ini membahas bagaimana mengkodekan data informasi menggunakan algoritma *Base64* setelah proses kompresi data. Hasilnya adalah ruang penyimpanan data informasi *QR Code* meningkat hingga 52% dengan algoritma kompresi data *GZip* dan *Base64.* [7]

Penelitian terakhir membahas tentang peningkatan data informasi yang disimpan pada *QR Code* dengan metode kompresi data dan *multiplexing*. Data

informasi yang sudah diubah menjadi *QR Code* akan dibagi menjadi 5 bagian, lalu disatukan kembali menjadi *QR Code* tunggal. Hasilnya daya tampung data infromasi *QR Code* mampu ditingkatkan. [8]

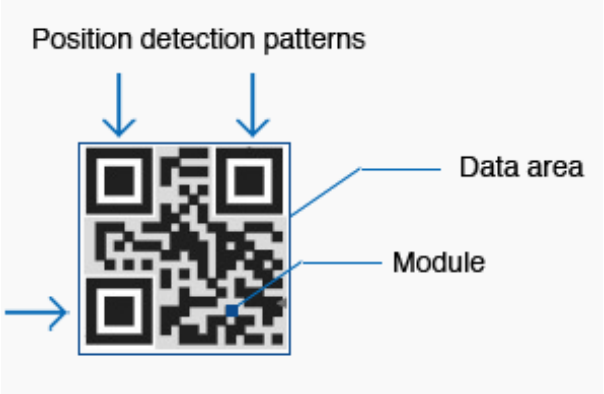
2.2 *QR Code*

QR Code merupakan bentuk 2D dari *barcode* yang dikembangkan oleh *Denso Wave Incorporation,* sebuah perusahaan otomotif asal jepang pada tahun 1994. Sejak awal diciptakan, hingga sekarang *QR Code* sudah memiliki 40 versi yang masing-masing menunjukkan jumlah data yang mampu disimpan. Setiap versi dari *QR Code* memiliki kapasitas data maksimum yang sesuai dengan jumlah data yang disimpan, tipe karakter, dan tingkat koreksi kesalahannya. Dengan kata lain, meningkatnya jumlah data maka akan lebih banyak modul yang dibuat dan menghasilkan simbol *QR Code* yang lebih besar.



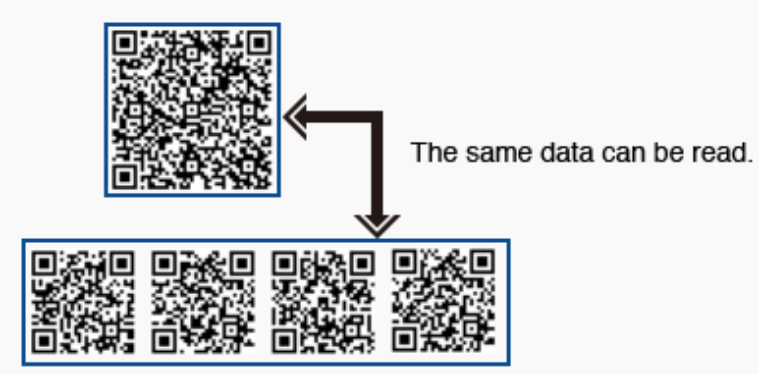
Gambar II.1 Versi *QR Code*

Keunggulan lainnya dari *QR Code* ini adalah dapat menyimpan karakter yang cukup banyak seperti Numerik, Alphanumerik, Biner, hingga karakter khusus seperti *Kanji* dan *Kana*. *QR Code* juga masih mampu terbaca walaupun rusak atau kotor, dengan maksimum koreksi *error*nya adalah 30%. Selain itu, *QR Code* mampu dibaca secara *omni-directional (360°)* secara cepat. *Omni-directional* mampu dilakukan karena *QR Code* memiliki symbol pada tiga sudut yang berbeda. Dengan pola deteksi seperti ini, menjamin untuk pembacaan dengan kecepatan tinggi dan stabil.



Gambar II.2 Letak dari *direction pattern*, *data area*, dan *module*

Untuk mengatasi sulitnya pembacaan *QR Code* karena data yang disimpan cukup banyak adalah dengan cara membaginya menjadi beberapa bagian dengan cara direkonstruksi menjadi *QR Code* baru.



Gambar II.3 *QR Code* mampu dibagi menjadi beberapa bagian

Proses mengubah data masukan menjadi gambar *QR Code* merupakan proses yang panjang. Ada sekitar tujuh langkah yang harus dilalui data sebelum diubah menjadi *QR Code* [1]. Langkah-langkah tersebut adalah sebagai berikut:

1. *Analisis Data*

Proses ini dilakukan agar dapat menentukan mode paling optimal yang bisa digunakan. Secara umum ada empat mode yang sering digunakan, yaitu: numerik, alfanumerik, *byte*, dan *kanji*. Setiap mode memiliki metode yang berbeda untuk mengubah teks menjadi bit sesingkat mungkin untuk tipe data tersebut.

2. *Data Encoding*

Data *encoding* dilakukan untuk menyandikan teks menjadi *string bit* ditambah dengan *error correction level (ECL). ECL* ini digunakan untuk mengecek apakah data yang dibaca sudah benar dan mengkoreksi jika terdapat kode yang *error*. Ada empat level dari *ECL* ini, yaitu: *Low(L), Medium(M), Quartile(Q), High(H).* Masing-masing level memiliki kemampuan mengkoreksi *error* yang berbeda. Semakin tinggi levelnya maka semakin banyak *error correction codeword* yang disisipkan.

3. *Error Correction Coding*

Error correction coding memungkinkan pemindai *QR Code* untuk mendeteksi dan memperbaiki kesalahan dalam *QR Code*.

4. *Structur Final Message*

QR Code yang lebih besar mengharuskan untuk memecah data *codeword* menjadi blok-blok yang lebih kecil, dan menghasilkan *error correction codeword* secara terpisah untuk setiap blok. Ketika hal ini terjadi, blok data dan kode kesalahan koreksi harus disisipkan sesuai dengan spesifikasi *QR Code*.

5. *Module Placement in Matrix*

Pada langkah sebelumnya (*structur final message*), data dan *error code correction* disatukan, dan *string final bit* diperoleh. Langkah selanjutnya adalah menempatkan mereka dalam matriks *QR Code* bersama dengan pola fungsi yang diperlukan. Pola fungsi adalah elemen non-data dari *QR Code* yang diperlukan oleh spesifikasi *QR Code*, seperti tiga *finder pattern* pada sudut-sudut matriks *QR Code*.

6. *Data Masking*

Setelah modul ditempatkan dalam matriks, *masking pattern* terbaik harus ditentukan. *Masking pattern* mengubah antara modul yang gelap dan yang terang menurut aturan yang sudah ditentukan. Tujuannya adalah untuk memudahkan pembacaan *QR Code*.

7. Format dan Informasi Versi

Langkah terakhir untuk membuat *QR Code* adalah membuat string format dan versi, lalu menempatkannya di lokasi yang benar dalam *QR Code*. Penyisipan format dan informasi versi dilakukan untuk memudahkan pemindai dalam melakukan pembacaan *QR Code* [9].

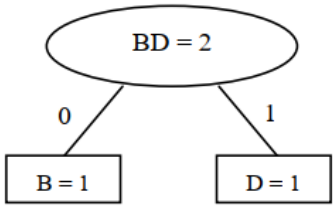
2.3 Algoritma Kompresi Data *Huffman*

Kompresi data merupakan cara untuk mereduksi jumlah *bit* data yang di representasikan. Saat ini kompresi digunakan secara luas untuk berbagai kebutuhan seperti komunikasi dan multimedia (video, musik, gambar). Dengan bertambahnya aktifitas manusia dalam mendigitalkan *file* fisik menjadi *softfile* maka akan bertambah pula jumlah *bit* data yang dihasilkan. Walaupun perangkat keras untuk ruang penyimpanan selalu ditingkatkan teknologi dan kapasitasnya, ini tidak cukup mampu untuk menampung jumlah data yang ada saat ini [10]. Saat ini ada dua teknik kompresi data, yaitu *lossy* dan *lossless*. Kompresi *lossless* mereduksi *bit* dengan cara mengidentifikasi dan mengeliminasi data yang redundansi. Sedangkan teknik kompresi *lossy* mengurangi data *bit* dengan mengidentifikasi informasi yang kurang penting lalu menghapusnya [6].

Algoritma *Huffman* merupakan salah satu metode kompresi data *lossless* yang popular untuk meng-kompresi data *text*. Algoritma ini diciptakan oleh David *Huffman* pada tahun 1952 dengan cara menggunakan prinsip yang sama seperti sandi morse. Proses *encoding* yaitu cara untuk mengubah setiap karakter menjadi beberapa seri bit, dimana karakter yang sering muncul akan diberikan kode dengan kode bit yang pendek, sedangkan karakter yang jarang muncul akan diberi kode dengan kode bit panjang. Algoritma *Huffman* menggunakan metode statis dalam proses *encoding*-nya. Metode statis yaitu metode yang menggunakan kode *map* yang sama namun urutan penampilan karakternya bisa berubah-ubah. Metode ini membutuhkan dua proses, yaitu yang pertama adalah menghitung frekuensi setiap karakter untuk menentukan kode *map*, lalu yang kedua konversi kedalam kumpulan kode biner [11].

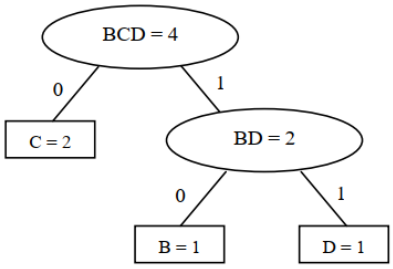
Dalam proses kompresi data *Huffman* ada empat fase yang akan dikerjakan. Misalnya kode ASCII string "ABACCDA" yang berukuran 56 bit atau 7 *byte*, karena masing-masing karakter menempati 8 bit atau 1 *byte* akan di kompresi dengan algoritma *Huffman*, langkah-langkahnya adalah sebagai berikut:

1. Baca semua karakter di dalam data untuk menghitung frekuensi kemunculan setiap karakter, lalu urutkan dari yang paling kecil frekuensinya. Dari contoh di atas, maka hasilnya adalah : A=3 C=2 B=1 D=1.
2. Gabungkan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah akar. Akar tersebut akan mempunyai frekuensi yang merupakan jumlah dari dua frekuensi pohon penyusunnya.



Gambar II.4 Langkah pertama penyusunan pohon *huffman.*

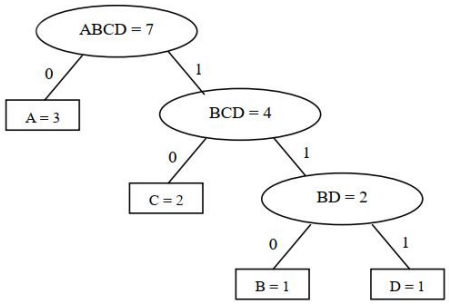
3. Ulangi langkah kedua hingga menyisakan satu buah pohon *Huffman*. Agar pemilihan dua pohon yang akan digabungkan berlangsung cepat, maka semua pohon yang ada selalu terurut menaik berdasarkan frekuensi.



Gambar *II.5* Langkah kedua penyusunan pohon *huffman.*

4. Berikan kode pada setiap karakter dalam pohon *Huffman* dengan susunan bit biner. Aturannya adalah bit "1" akan selalu ditempatkan pada sisi kanan, dan bit "0" ditempatkan pada sisi kiri. Dengan begitu setiap karakter akan

memiliki alamat bit binernya sendiri. Misalnya karakter B akan memiliki alamat bit biner “110”, sedangkan karakter D akan beralamat “111”.



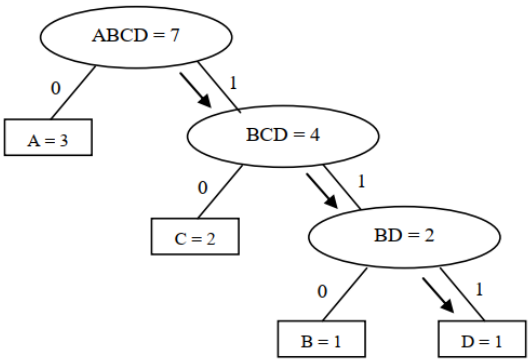
Gambar II.6 Langkah ketiga penyusunan pohon *huffman.*

Dari pohon biner di atas, maka string “ABACCDA” yang berukuran 56 bit(7 *byte*) akan digantikan oleh susunan bit biner 0110010101110 yang hanya akan berukuran 13 bit (~ 2 *byte*). Pada tabel II.2 akan dijelaskan lebih lanjut.

Tabel II.1 Kode karakter berdasarkan pohon biner.

No	Data Sampel	Ukuran (bytes)
1	A	0
2	B	110
3	C	10
4	D	111

Dalam proses dekompresi, atau mengembalikan data yang sebelumnya sudah dimampatkan dengan cara menelusuri pohon biner tersebut mulai dari akar. Misalnya jika kita akan mendekompresi string “111”, maka telusuri dari akar hingga diketahui bahwa string tersebut bernilai D (Gambar II.7).



Gambar II.7 Langkah keempat penyusunan pohon *huffman.*

Selain dengan menelusuri pohon biner, dapat dilihat pada Tabel II.2 yang sudah dibuat. Prosesnya adalah baca terlebih dahulu susunan biner yang dikompresi, dalam hal ini adalah 0110110101110. Bit pertama dari sususan tersebut adalah 0 yang merupakan pengganti dari karakter A, lalu baca bit berikutnya, yaitu 1, jika tidak ada karakter yang diganti maka baca kembali bit berikutnya hingga menjadi 11, jika 11 juga tidak ada maka baca kembali bit setelahnya hingga menjadi 110 yang merupakan karakter pengganti dari B. Lakukan pembacaan bit sampai bit terakhir sehingga proses dekompresi akan menghasilkan string semula yaitu "ABACCDA" [12].