目 次

第1章　序論

QR コード [1] は、1994 年に株式会社デンソーウェーブが開発した二次元バーコード であり、食品や交通分野など多方面の分野で利用されている．白と黒の正方形のモジュールから構成され、ランダムな見た目で表される．一般的な QR コードはデザイン性を考慮していないが、一方で広告、サービス業界ではデザイン性を考慮した QR コードが求められている．デザイン性を考慮した QR コードでは、一定のルールから QR コードを変更することによって、QR コード上にロゴ画像（以後、目的画像と述べる）を埋め込んだものがある．このような QR コードを Aesthetic QR コード [3] という．

Aesthetic QR コードの研究は大きく三種類に分けることができる．

1. QR コードの一部に目的画像を埋め込む方法.
2. 画像のヒストグラムを考慮して目的画像を埋め込む方法 [2].
3. RS 符号のパディングビットを考慮して目的画像を埋め込む方法 [3].

本研究は、上に述べた三つ目に分類される RS 符号のパディングビットを考慮して目的画像を埋め込む方法を考察し、目的画像に近い Aesthetic QR コードを自動生成する方法について検討する．Aesthetic QR コードを自動生成する手法として、本研究では、Kuribayashi らの論文 [3] で提案されているランダム手法のアルゴリズムを用い、Aesthetic QR コードを生成するソフトウェアを開発することを研究の目的とする．

以下、第 2 章では QR コードを構成する Reed-Solomon 符号と QR コードの概要について述べ、第 3 章では本研究のベースとなっている Kuribayashi らの論文 [3] のランダム法について述べる. 第 4 章では実験により得られた結果について述べる. 第 5 章では結論と今後の課題について述べる.

第2章　QRコード

2.1　Reed-Solomon 符号

Reed-Solomon 符号 (以下、RS 符号) は誤り訂正符号の一つであり、有限体や拡大体上で実装される. その誤り訂正能力は高く QR コードでも応用されている. パリティ部を用いることで誤りを検知でき、決められた数以下のノイズであれば誤りを訂正できる.

拡大体 $GF(p^j)$ 上において、符号のデータは複数のビットを 1 つのデータ単位として扱い、このデータ単位のことをシンボルと呼ぶ.

RS 符号は RS 符号の符号長を n シンボル、シンボル・エラー訂正最大数を d 個、データ長を k シンボルと置くとき、(n,k) 符号と呼ぶ. RS 符号のデータ長 k シンボルはデータ部と呼ぶ. データ長 $n-k$ シンボルはパリティ部と呼び、誤り訂正に使われる.

RS 符号のデータ部は固定長である．QR コードに入れる文字列が既定の数より少ない場合、データ部の係数を所定の数に合わせるためパディング（埋め草コード）を付加する．入力データの長さを $\hat{k}$ と表す.
入力データを表すシンボルを

$$\alpha_1, \cdots, \alpha_{\hat{k}} \tag{2.1}$$

である．$\hat{k} < k$ の時、RS 符号のデータ部の係数を k 個に合わせるため、パディングを付加する．これにより、RS 符号のデータ部のシンボルは

$$\alpha_1, \cdots, \alpha_{\hat{k}}, \alpha_{\hat{k}+1}, \cdots, \alpha_k$$

とされる．ここで、パディングは

$$\alpha_{\hat{k}+1}, \cdots, \alpha_k \tag{2.2}$$

とである.

RS 符号の全体図を図 2.1 に示す.

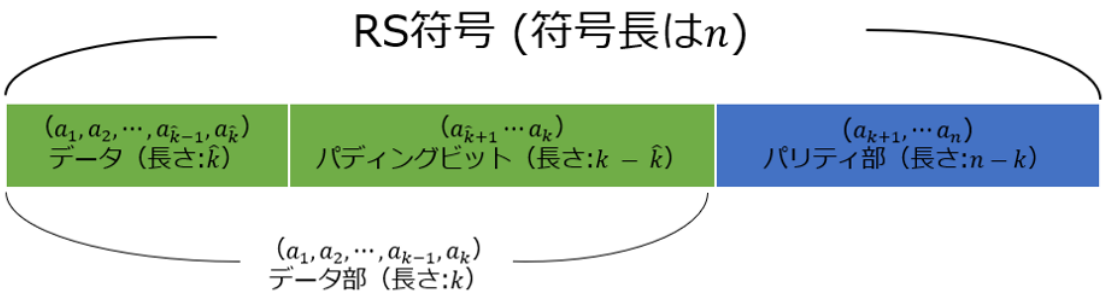

図 2.1　RS 符号の全体図

パディングビットは QR コードに入力するデータとは関係ないため、自由に変更できる. パディングビットを変更することにより、RS 符号を形成する多項式の係数は変わる. そのため、RS 符号化で得られる符号の値も変化する.

計算は有限体や拡大体で行われる. 符号で用いられる値は原始多項式の元 α で定義される. 符号を特定するには n, k 以外に、生成多項式 [4] [5] と呼ばれる $n-k$ 次多項式 $g(x)$ を選ぶ必要がある.

QR コードでは、

$$g(x) = (x-1)(x-\alpha)\cdots(x-\alpha^{n-k-1}) = \prod_{i=0}^{n-k-1}(x-\alpha^i) \tag{2.3}$$

であり、式 2.3 の $g(x)$ を展開した多項式を

$$g(x) = g_1 x^{n-k} + g_2 x^{n-k-1} + \cdots + g_{n-k+1},\ g_1 = 1 \tag{2.4}$$

とする. その係数列 $g_1 = 1, g_2, \cdots, g_{n-k+1}$ から定まる $k \times n$ 行列

$$G_0 = \begin{bmatrix} 1 & g_2 & \cdots & g_{n-k+1} & 0 & \cdots & \cdots & 0 \\ 0 & 1 & g_2 & \cdots & g_{n-k+1} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & g_2 & \cdots & g_{n-k+1} & 0 \\ 0 & \cdots & \cdots & 0 & 1 & g_2 & \cdots & g_{n-k+1} \end{bmatrix}$$

を生成行列と呼ぶ. 組織符号を得るには、行列 G_0 を”掃き出し法”によって $G = [I_k, P]$ (I_kは $k \times k$ 単位行列) という標準形に変形する.

RS 符号は次のようにして得ることができる. データ u に対する符号語 v は標準形の生成行列 $G = [I_k, P]$ により、

$$v = uG \tag{2.5}$$

として表現することができる.

2.2　QR コードの概要

QR コードの構成要素の最小単位は白と黒で表されるモジュールであり、各モジュールには単一ビット値が割り当てられる. QR コードのサイズはバージョンによって決定され、そのバージョン (v) は 1 ~ 40 である. 1 型は、21 × 21 モジュール、2 型は、25 × 25 モジュール、というように、型番が一つ上がるごとに一辺につき 4 モジュールずつ増加し、40 型は、177 × 177 モジュールとなる. したがって、バージョン v は $(17 + 4v) \times (17 + 4v)$ モジュールである.

図 2.2 に QR コードの構成要素を表す.

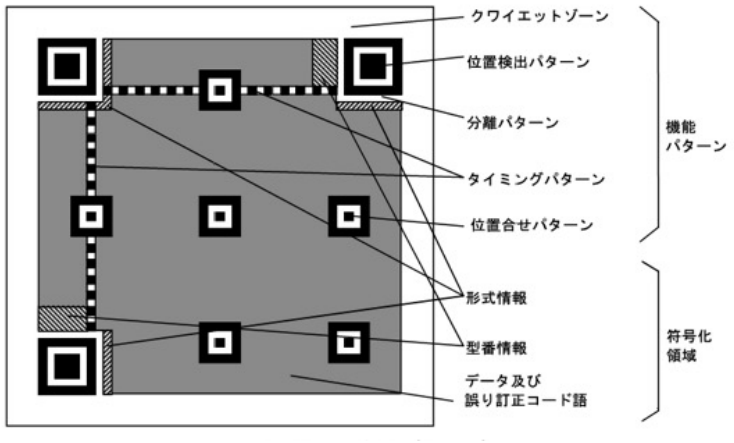

図 2.2　QR コードシンボルの構造 [6]

QR コードは、QR コード上にある符号化されたデータを正確に認識するために機能パターンを持っている. 機能パターンは主に 3 つの構成要素から成り立っており、それぞれ位置検出パターン、位置合わせパターン、タイミングパターンと呼ばれる.

位置検出パターンは QR コードの左上、左下、右上の角にある 3 つの正方形のブロックである. それらの境界を明確にするために、形式情報との間に白いモジュールを置く. これを

分離パターンと呼ぶ.

位置合わせパターンは小さな正方形のブロックで、位置検出パターンの垂直・平行座標に関係する位置に置く. バージョンによっては付加しない場合もあり、バージョン 1 には存在しない.

タイミングパターンは左上の位置検出パターンから右上の位置検出パターンへと、左上の位置検出パターンから左下の位置検出パターンへの白黒が交互に並ぶ 2 つのラインのことである.

QR コードのデータビットは、QR コードの右下から始まり、2 モジュール幅の列上に配置する．列が最上部に達すると、次の 2 モジュール列は右端から始まり、下方向へ続く．現在の列が端に達すると、次の 2 モジュールの列に移動して方向を変更する．データビットは機能パターン（位置検出パターン、タイミングパターン、位置合わせパターン）の位置では、次のモジュールへ配置される.

上方向のデータビットの配置は表 2.1 に、下方向のデータビットの配置は表 2.2 に示す.

表 2.1　上方向のビット配列

0	1
2	3
4	5
6	7

表 2.2　下方向のビット配列

6	7
4	5
2	3
0	1

QR コードは、誤り訂正に RS 符号を使用し、その能力は L、M、Q、H の 4 つのレベルに昇順で分類される. 各誤り訂正レベルは QR コード内の全シンボルの約 7%、約 15%、約 25%、約 30% までのシンボルを訂正することができる. それぞれを表 2.3 に示す.

表 2.3　誤り訂正レベル

レベル	L	M	Q	H
誤り訂正能力	約 7%	約 15%	約 25%	約 30%

第3章　Aesthetic QRコード

本章では、QR コード上に配置された二値配列が埋め込む目的画像のモジュールパターンと類似した、より埋め込む画像に近い QR コードを得るために用いた Kuribayashi らの論文 [3] のランダム法について述べる.

Aesthetic QR コードを作成するにあたり、本研究では QR コードに埋め込む画像を目的画像と定義する.

Aesthetic QR コードの例を図 3.1 に示す.

図 3.1　Asethetic QR コード例

3.1　色変換手法

ランダム法を使用する際に目的画像を二値化して QR コードとのハミング距離を測り、最もハミング距離の小さい QR コードを用いて Aesthetic QR コードを作る. 目的画像を二値化画像にする際に閾値を指定するが、その閾値を Algorithm 1 に示す色変換手法で決定する.

閾値を用いて目的画像から目的画像の二値行列を作成するアルゴリズムを Algorithm 2 に示す.

Algorithm 1 論文 [3] の色変換手法

入力: サイズ $L \times L$ の目的画像

出力: 閾値 $\overline{Y}$

方法:
1. 入力画像の大きさは、QR コードのバージョン v と同じサイズに予め変更しておく．RGB 色成分は YUV 色成分に変換され、輝度 (Y) 成分 $Y_{i,j}$,$(1 \leq i, j \leq L)$ が得られる．
2. その中心の正方形 (元の画像サイズの $\frac{1}{4}$) の値の平均 $\overline{Y}$ が計算する．

$$\overline{Y} = \frac{4}{L^2} \sum_{i=\frac{L}{4}}^{\frac{3L}{4}-1} \sum_{j=\frac{L}{4}}^{\frac{3L}{4}-1} Y_{i,j} \tag{3.1}$$

Algorithm 2 目的画像に対する二値行列の生成

入力: サイズ $L \times L$ の目的画像の輝度 (Y) 成分 $Y_{i,j}(1 \leq i, j \leq L)$、閾値 $\overline{Y}$

出力: 二値行列 $B_{i,j}$

方法:
1. 目的画像を二値化する際、二値行列 $B_{i,j}$ は、以下の規則によって決定される．

$$B_{i,j} = \begin{cases} 1 & Y_{i,j} > \overline{Y} \\ 0 & otherwise \end{cases} \tag{3.2}$$

3.2　ランダム法

ランダム法では、QR コードと目的画像の画像サイズを等いものとして、QR コードの 1 モジュールと目的画像の 1 画素を対応させる．目的画像と QR コードとの違いを導き出すための手順として、目的画像の二値化を行い、QR コードと二値行列とのハミング距離を取ることを行う．

RS 符号化を行う際、掃き出し法により G を $(I|P)$ の形に変形せず、I の後半の列 (パディング) を P のどこかの列に入るように掃き出し法を行う．ランダム法では N 回 RS 符号を生成し、目的画像に最も近い QR コードを計算する．目的画像の二値化画像と生成された QR コードのハミング距離を取り、ハミング距離が最小になる QR コードを使って Aesthetic QR コードを作成する．

パディングは長さ $n-k$ のパリティ部のどこかに置くこととなる. パディングの長さは $k-\hat{k}$ で与えられるため,$C(n-\hat{k}, k+\hat{k})$ のパディングを置く場所についての組み合わせが存在する. 試行回数は $N \leq C(n-\hat{k}, k-\hat{k})$ を満たす値になる．

本研究では QR コードの中でバージョン 1 の QR コードを用いた. バージョン 1 の QR コードを作成する手順を以下に説明する．

Algorithm 3 ランダム法を用いたバージョン 1 の Aesthetic QR コード

入力: バージョン 1 の QR コードに入る範囲内の文字列、サイズ 21×21 の目的画像

出力: サイズ 21×21 のバージョン 1 の Aesthetic QR コード

方法:

1. 目的画像の画素値を QR コードのモジュールに割り当て、Algorithm2 で決まった閾値 $\overline{Y}$ でモジュールを二値化し、二値行列 $B_{i,j}$ を作成する．
2. $B_{i,j}$ に所定のマスキングパターンを作用させる．
3. 式 (2.2) の α_t $(\hat{k}+1 \leq x_t \leq n)$ を変化させることにより、RS 符号を見つける．以後この手順を N 回繰り返すことにより、$B_{i,j}$ とのハミング距離が最小となる RS 符号を見つける．一定の試行回数終了後、QR コード上の RS 符号は、最良の RS 符号に置き換える．
4. マスク処理前である QR コードの各モジュールに対して、所定のマスクパターンを適用する．
5. 目的画像と QR コードをかさねて Aethetic QR コードを作成する．

第4章　バージョン1のAesthetic QRコードのソフトウェア実装と評価

4.1　ソフトウェアの開発環境

ソフトウェアの実装に使用した PC 環境、言語を以下に示す.

- ソフトウェア実装環境
  - CPU：Intel(R)Core(TM) i7-7700 CPU @ 3.60GHz 3.60GHz
  - OS：Windows 10 pro
  - 実装 RAM：16.0GB
- 開発環境
  - Java：Java 1.8.0_241
  - Eclipse：Eclipse 4.11.0
  - Maple：Maple 2017.3

本研究で開発したソフトウェアが対応するバージョン 1 の QR コードのパラメータを以下に示す.

- QR コード
  - RS 符号：(26,16) 符号
  - 入力文字：FUKUDA
  - バージョン (v)：1
  - マスクパターン：001
  - 誤り訂正レベル：M

また、バージョン 1 の QR コードの構造を図 4.1 に示す.

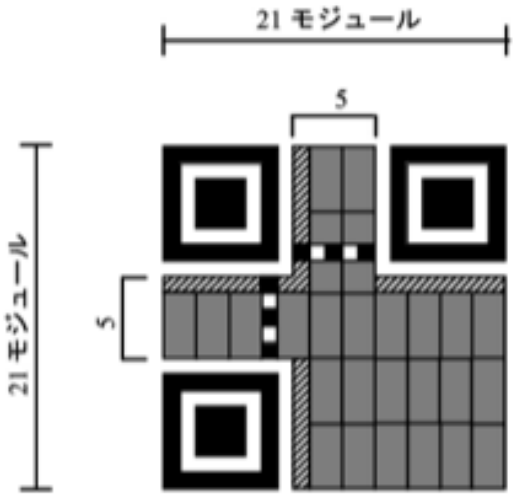

図 4.1　バージョン 1 の QR コードの構造

4.2　開発したソフトウェア

RS 符号化は有限体の実装の容易な数式処理システム Maple（以下 Maple）、QR コードの生成は画像が扱いやすい Java を用いる. ソフトウェアの実行手順を以下に示す.

1. 文字列を Maple で作成したプログラムに入力し、RS 符号が得られる.
2. RS 符号と QR コードに含めたい目的画像を Java に入力し、Aesthetic QR コードが出力される.

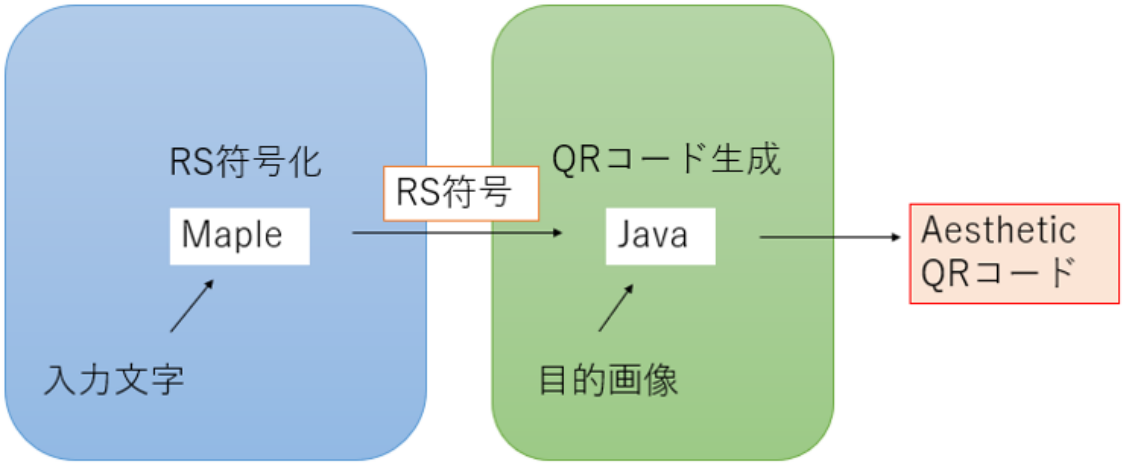

図 4.2　開発したソフトウェアの Aesthetic QR コードの作成までの流れ

4.3　評価

Aestetic QR コードの評価は目的画像の二値行列 $B_{i,j}$ と得られた QR コードの二値行列におけるハミング距離、$PSNR$(ピーク信号対雑音比) で行う.

ハミング距離は Kuribayashi らの論文 [3] で使われている評価であり、小さければ RS 符号と目的画像の二値化行列が近いことを指す. PSNR は画像の劣化具合を評価するのに一般的に使われている尺度であり、大きければ劣化が少ないことを指す.

4.3.1　最小ハミング距離

2 つの符号語つの符号語 $a = (a_1, a_2, \cdots, a_n)$ と $b = (b_1, b_2, \cdots, b_n)$ で対応するビット (桁）で値 (0 または 1) が異なっているビット (桁) の数をハミング距離と言い、記号で $d(a, b)$ と書く. その中でも一番ハミング距離が小さいものを最小ハミング距離と呼ぶ.

ハミング距離は 2 つの符号語 $a = (a_1, a_2, \cdots, a_n)$ と $b = (b_1, b_2, \cdots, b_n)$ に対して以下の式で定義される.

$$d(a, b) = \sum_{i=1}^{n} (a_i + b_i) \pmod 2 \tag{4.1}$$

4.3.2　$PSNR$(ピーク信号雑音比)

$PSNR$ は画質の再現性に影響を与える信号がとりえる最大の輝度と劣化をもたらすノイズの比率を示したものである. $PSNR$ は以下の式で定義される.

$$PSNR = 10 \log 10 \frac{MAX_I^2}{MSE} \tag{4.2}$$

$$= 20 \log 10 \frac{MAX_I}{\sqrt{MSE}} \tag{4.3}$$

MAX_I は画像がとりうる最大ピクセルを表す. ピクセルの 1 サンプルが 8 ビットで表現されている場合、MAX_I は 255 である. MSE(平均二乗誤差) は元画像と処理画像との差の 2 乗誤差である.MSE が小さければ小さいほど元画像に近い画像である. MSE は $M \times N$ の 2 つの画像 I、K において、以下の式で定義される.

$$MSE = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I(i, j) + K(i, j)]^2 \tag{4.4}$$

4.4　実験結果

本実験で QR コードに挿入する画像は図 4.3 の目的画像を使用した. バージョン 1 の QR コードで用いられる RS 符号は $n = 26, k = 16$ である. 本実験では 6 文字の長さの文字列を用いているので $\hat{k} = 6$ である. 従って $N \leq C(n - \hat{k}, k - \hat{k}) = C(20, 10) = 184756$ である. 本実験では、N を 10000 までとして実験を行った. ランダム法の試行回数が $N = 1, 10, 100, 1000, 10000$ の場合の Aestheic QR コードを図 4.5~ 図 4.9 に表す.

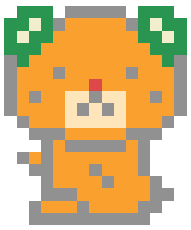

図 4.3　目的画像

図 4.4　目的画像に QR コードを張り付けただけの QR コード

図 4.5　$N = 1$

図 4.6　$N = 10$

図 4.7　$N = 100$

図 4.8　$N = 1000$

図 4.9　$N = 10000$

また、表 4.1 に各試行回数における Aesthetic QR コードのハミング距離と $PSNR$ 値、RS 符号の生成にかかった計算時間を表す.

表 4.1　結果のまとめ

生成した回数 (N)	1	10	100	1000	10000
ハミング距離	83	69	67	53	47
$PSNR$ 値 (dB)	31.50	31.99	32.07	32.82	33.04
N 回の RS 符号の計算時間（秒）	0.06	0.4	4.2	42.0	425.5

表 4.1 を見ると、試行回数を増やしていくたびに評価の値が向上しており、より良い Aesthetic QR コードが生成されている.

しかし、試行回数 10000 回では RS 符号を生成するのに約 7 分も時間がかかっており、あまり実用的ではない. Aesthetic QR コードの計算時間を短縮するには、アルゴリズムの改良や、並列処理などが考えられるかもしれないが、今後の課題として検討する.

第5章　結論

本研究ではランダム法によって生成されるバージョン 1 の Aesthetic QR コードのソフトウェア実装を行った. 開発環境として、有限体の実装が容易な数式処理システム Maple と画像を扱いやすい Java を利用しており、QR コードで符号化する文字列と目的画像を与えることで Aesthetic QR コードが得られる環境を整えることができた. 今後の課題として、

- Aesthetic QR コードの計算時間の短縮
- Java のプログラムと Maple のプログラムの連結手法の検討
- 生成可能な Aesthetic QR コードのバージョンの追加

などが挙げられる.

謝辞

本研究に際して、日々、様々なご指導をいただきました甲斐博准教授に心より感謝致します. 共同研究に際して、ご指導いただきました森井昌克教授に深謝いたします. そして、本研究に際してご審査いただきました伊藤宏教授、稲元勉講師に感謝の意を表します. 最後に日頃から助言や励ましをいただきました諸先輩方、並びに同研究室の皆様に深く御礼を申し上げます.

参考文献

［1］ QR code.com. http://www.qrcode.com/en.

［2］ Visualead. http://www.visualead.com.

［3］ M. Kuribayashi and M . Morii ”Aesthetic QR Code Based on Modified Systematic Encoding Function”, IEICE transactions on information and systems ,VOL.E100–D, NO.1,pp.42-51,2017.

［4］ 池田和興, 例題が語る符号理論, 社共立出版, 2007 年

［5］ J. Justesen and T. Hoholdt ”A Course In Error-Correctiong Codes”, European Mathematical Society Publishing House, 2004.

［6］ JIS X0510. http://www.jisc.go.jp/app/pager?id=2738494.

付 録A　プログラムリスト

以下のファイルは本研究のために作成したプロフラムファイルである. 以下にソースコードを示す.

[Main.java]

```
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.regex.Pattern;
import javax.imageio.ImageIO;

public class Main {
  public static int a(int c) {
    return c >>> 24;
  }
  public static int r(int c) {
    return c >> 16 & 0xff;
  }
  public static int g(int c) {
    return c >> 8 & 0xff;
  }
  public static int b(int c) {
    return c & 0xff;
  }
  public static int rgb(int r, int g, int b) {
    return 0xff000000 | r << 16 | g << 8 | b;
  }
  public static int argb(int a, int r, int g, int b) {
    return a << 24 | r << 16 | g << 8 | b;
  }
  public static void main(String[] args) throws IOException {
    // 符号全体(ここをMapleとうまく繋げたい)(今は手動で
        Mapleで作られたものを持ってきている)
    //RS符号の入力
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);

    System.out.println("RS符号を入力してください(例 11101, 1011,)");

    String str = null;
```

```
37      try {
38        str = br.readLine();
39        br.close();
40      } catch (IOException e) {
41        e.printStackTrace();
42      }
43
44      Pattern p = Pattern.compile("[,\\s]+");
45      String[] str_1 = p.split(str);
46
47      int[] bit8_array = new int[str_1.length];
48
49      for (int i = 0; i < str_1.length; i++) {
50        try {
51          bit8_array[i] = Integer.parseInt(str_1[i]);
52        } catch (NumberFormatException e) {
53          System.out.println("正しいRS符号を入力してください");
54          System.out.println("プログラムを終了します。");
55          return;
56        }
57      }
58
59      //RS符号の確認
60      System.out.println("正しくRS符号が読み込めました");
61
62      // 10進符号→2進符号に変換
63      for (int i = 0; i < bit8_array.length; i++) {
64        bit8_array[i] = Integer.parseInt(String.valueOf(bit8_array[i]), 2);
65      }
66
67      // ファイル名
68      String inname = "C:\\Users\\fukuda\\Desktop\\QRコード関連みきゃんの\\2値使ってる
            の化\\\\101_マスクパターン.png";
69      String inname_2 = "C:\\Users\\fukuda\\Desktop\\QRコード関連みきゃんの\\2値使ってるのみきゃ
            ん化\\\\.png";
70      String outname = "C:\\Users\\fukuda\\Desktop\\QRコード関連みきゃんの\\2値使ってるのじっけん
            んn化\\\\.png";
71      // 画像格納クラス
72
73      // イメージをBufferedImageへ読みこむ。
74      // 以下イメージの操作はBufferedImageを利用して行う。
75      BufferedImage image = ImageIO.read(new File(inname));
76      BufferedImage image_2 = ImageIO.read(new File(inname_2));
77
78      // 元イメージの幅、高さを取得。
79      int w = image.getWidth();
80      int h = image.getHeight();
81
82      if (w != 21 || h != 21) {
83        System.out.println("画像サイズが違います！");
84        return;
85      }
```

```
86
87      // 書き込み（スパゲッティコード）(最悪だが、今は一時的にこれで)（直したい）
88      int bit8;
89      write_module(image, bit8_array[0], 19, 17, false);
90      write_module(image, bit8_array[1], 19, 13, false);
91      write_module(image, bit8_array[2], 19, 9, false);
92
93      write_module(image, bit8_array[3], 17, 12, true);
94      write_module(image, bit8_array[4], 17, 16, true);
95      write_module(image, bit8_array[5], 17, 20, true);
96
97      write_module(image, bit8_array[6], 15, 17, false);
98      write_module(image, bit8_array[7], 15, 13, false);
99      write_module(image, bit8_array[8], 15, 9, false);
100
101     write_module(image, bit8_array[9], 13, 12, true);
102     write_module(image, bit8_array[10], 13, 16, true);
103     write_module(image, bit8_array[11], 13, 20, true);
104
105     write_module(image, bit8_array[12], 11, 17, false);
106     write_module(image, bit8_array[13], 11, 13, false);
107     write_module(image, bit8_array[14], 11, 9, false);
108     bit8 = bit8_array[15];
109     if (bit8 % 2 != 1)
110       image.setRGB(11, 4, 0x000000);
111     bit8 /= 2;
112     if (bit8 % 2 != 1)
113       image.setRGB(12, 4, 0x000000);
114     bit8 /= 2;
115     if (bit8 % 2 == 1)
116       image.setRGB(11, 5, 0x000000);
117     bit8 /= 2;
118     if (bit8 % 2 == 1)
119       image.setRGB(12, 5, 0x000000);
120     bit8 /= 2;
121     if (bit8 % 2 == 1)
122       image.setRGB(11, 7, 0x000000);
123     bit8 /= 2;
124     if (bit8 % 2 == 1)
125       image.setRGB(12, 7, 0x000000);
126     bit8 /= 2;
127     if (bit8 % 2 != 1)
128       image.setRGB(11, 8, 0x000000);
129     bit8 /= 2;
130     if (bit8 % 2 != 1)
131       image.setRGB(12, 8, 0x000000);
132     bit8 /= 2;
133     write_module(image, bit8_array[16], 11, 0, false);
134
135     write_module(image, bit8_array[17], 9, 3, true);
136     bit8 = bit8_array[18];
```

```
137       if (bit8 % 2 != 1)
138         image.setRGB(9, 8, 0x000000);
139       bit8 /= 2;
140       if (bit8 % 2 != 1)
141         image.setRGB(10, 8, 0x000000);
142       bit8 /= 2;
143       if (bit8 % 2 == 1)
144         image.setRGB(9, 7, 0x000000);
145       bit8 /= 2;
146       if (bit8 % 2 == 1)
147         image.setRGB(10, 7, 0x000000);
148       bit8 /= 2;
149       if (bit8 % 2 == 1)
150         image.setRGB(9, 5, 0x000000);
151       bit8 /= 2;
152       if (bit8 % 2 == 1)
153         image.setRGB(10, 5, 0x000000);
154       bit8 /= 2;
155       if (bit8 % 2 != 1)
156         image.setRGB(9, 4, 0x000000);
157       bit8 /= 2;
158       if (bit8 % 2 != 1)
159         image.setRGB(10, 4, 0x000000);
160       bit8 /= 2;
161       write_module(image, bit8_array[19], 9, 12, true);
162       write_module(image, bit8_array[20], 9, 16, true);
163       write_module(image, bit8_array[21], 9, 20, true);
164
165       write_module(image, bit8_array[22], 7, 9, false);
166
167       write_module(image, bit8_array[23], 4, 12, true);
168
169       write_module(image, bit8_array[24], 2, 9, false);
170
171       write_module(image, bit8_array[25], 0, 12, true);
172
173       //目的画像にQRコードをかぶせる
174       QRcode_to_Mican(image, image_2, h, w);
175
176       // イメージをファイルに出力する
177       ImageIO.write(image, "png", new File(outname));
178
179       // 正常に終了
180       System.out.println("正常にAestheticQRコードが出来ました");
181     }
182
183     public static void write_module(BufferedImage writebuf, int bit8, int origin_x, int
          origin_y, boolean isUp) {
184       int sin_y;
185       sin_y = isUp ? -1 : 1;
186
```

```
187       for (int offset_y = 0; offset_y < 4; offset_y++) {
188         for (int offset_x = 0; offset_x < 2; offset_x++) {
189           if (((bit8 % 2) ^ ((origin_y + sin_y * offset_y) % 2)) == 0) {
190             writebuf.setRGB(origin_x + offset_x, origin_y + sin_y * offset_y, 0x000000);
191           }
192           bit8 /= 2;
193         }
194       }
195     }
196
197     //
198     public static void QRcode_to_Mican(BufferedImage Read_QRcode, BufferedImage
          Read_QRcode_2, int h, int w) {
199
200       for (int x = 0; x < h; x++) {
201         for (int y = 0; y < w; y++) {
202           int c = Read_QRcode.getRGB(x, y);
203           int c2 = Read_QRcode_2.getRGB(x, y);
204
205           // モノクロに変換
206           int mono = (int) (0.299 * r(c) + 0.587 * g(c) + 0.114 * b(c));
207           if (mono == 0) {
208             Read_QRcode.setRGB(x, y, 0x000000);//QRコードの部分（黒にする）
209
210           } else {
211             Read_QRcode.setRGB(x, y, c2);//目的画像の部分
212           }
213         }
214
215       }
216     }
217
218 }
```

[Binarization.java]

```java
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class Binarization {

  public static int a(int c) {
    return c >>> 24;
  }
  public static int r(int c) {
    return c >> 16 & 0xff;
  }
  public static int g(int c) {
    return c >> 8 & 0xff;
  }
  public static int b(int c) {
    return c & 0xff;
  }
  public static int rgb(int r, int g, int b) {
    return 0xff000000 | r << 16 | g << 8 | b;
  }
  public static int argb(int a, int r, int g, int b) {
    return a << 24 | r << 16 | g << 8 | b;
  }

  public static void main(String[] args) throws IOException {

    // ファイル名
    String inname = "C:\\Users\\fukuda\\Desktop\\QRコード関連みきゃんの\\2値使ってるのみきゃん化\\\\.png";
    String outname = "C:\\Users\\fukuda\\Desktop\\QRコード関連みきゃんの\\2値使ってるのみきゃん化\\\\_2値化.png";

    // 画像格納クラス

    // イメージをBufferedImageへ読みこむ。
    // 以下イメージの操作はBufferedImageを利用して行う。
    BufferedImage readImage = ImageIO.read(new File(inname));

    // モノクロに変換
    // 元イメージの幅、高さを取得。
    int w = readImage.getWidth();
    int h = readImage.getHeight();

    // 変換結果を書き込むBufferedImageを作成する。
    // サイズは元イメージと同じ幅、高さとする。
    BufferedImage write = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    //ヒストグラムの初期化
    int hist[] = new int[256];
```

```
50
51      // 1ピクセルづつ処理を行う
52      for (int y = 0; y < h; y++) {
53        for (int x = 0; x < w; x++) {
54          // ピクセル値を取得
55          int c = readImage.getRGB(x, y);
56          // 0.299や0.587といった値はモノクロ化の定数値
57          int mono = (int) (0.299 * r(c) + 0.587 * g(c) + 0.114 * b(c));
58          write.setRGB(x, y, rgb(mono, mono, mono));
59        }
60      }
61
62      //グレースケールのヒストグラムを作成
63      for (int y = 0; y < h; y++) {
64        for (int x = 0; x < w; x++) {
65          int c = readImage.getRGB(x, y);
66          int mono = (4899 * r(c) + 9617 * g(c) + 1868 * b(c) + 8192) >> 14;
67          hist[mono]++;
68        }
69      }
70
71      //閾値を設定（色変換手法）
72      double ave = 0;
73      for (int y = h / 4; y < h * 3 / 4; y++) {
74        for (int x = w / 4; x < w * 3 / 4; x++) {
75          int c = readImage.getRGB(x, y);
76          int mono = (int) (0.299 * r(c) + 0.587 * g(c) + 0.114 * b(c));
77          ave += mono;
78        }
79      }
80      ave *= 4;
81      ave /= h;
82      ave /= w;
83
84      //System.out.println(ave);（確認用）
85
86      //閾値を元に2値化する
87      for (int y = 0; y < h; y++) {
88        for (int x = 0; x < w; x++) {
89          int c = readImage.getRGB(x, y);
90          int mono = (int) (0.299 * r(c) + 0.587 * g(c) + 0.114 * b(c));
91          if (mono < ave) {
92            write.setRGB(x, y, 0x000000);//黒へ
93
94          } else {
95            write.setRGB(x, y, 0xFFFFFF);//白へ
96          }
97        }
98      }
99
100     // イメージをファイルに出力する
```

```
101       ImageIO.write(write, "png", new File(outname));
102       // 正常に終了
103       System.out.println("OK!");
104
105     }
106
107   }
```

[MSE_PSNR.java]

```
1
2  import java.awt.Color;
3  import java.awt.image.BufferedImage;
4  import java.io.File;
5  import java.io.IOException;
6  import javax.imageio.ImageIO;
7
8  public class MSE_PSNR {
9    public static void main(String[] args) {
10     try {
11       //比較する画像のパスを指定
12       String Comparison_1 = "C:\\Users\\fukuda\\Desktop\\QRコード関連みきゃんの\\2値使ってる
             のみきゃん化\\\\.png";
13       String Comparison_2 = "C:\\Users\\fukuda\\Desktop\\QRコード関連みきゃんの\\2値使ってる
             の出力化\\\\_10000.png";
14
15       //画像ファイルを読み込む
16       BufferedImage img_1 = ImageIO.read(new File(Comparison_1));
17       BufferedImage img_2 = ImageIO.read(new File(Comparison_2));
18
19       //もろもろの値の初期化
20       double sum = 0;
21       double ave_1 = 0;
22       double ave_2 = 0;
23       double max_1 = 0;
24       double max_2 = 0;
25       double min = 1000000;
26
27       //画像の縦、横の長さを取得
28       int h = img_1.getHeight();
29       int w = img_1.getWidth();
30
31       //MES値を計算
32       for (int i = 0; i < w; i++) {
33         for (int j = 0; j < h; j++) {
34           Color color_1 = new Color(img_1.getRGB(i, j));
35           Color color_2 = new Color(img_2.getRGB(i, j));
36           ave_1 = (color_1.getRed() + color_1.getGreen() + color_1.getBlue());
37           if (ave_1 > max_1) {
38             max_1 = ave_1;
39           }
40           ave_2 = (color_2.getRed() + color_2.getGreen() + color_2.getBlue());
41           if (ave_2 > max_2) {
42             max_2 = ave_2;
43           }
44           sum += Math.pow(color_1.getRed() - color_2.getRed(), 2)
45               + Math.pow(color_1.getGreen() - color_2.getGreen(), 2)
46               + Math.pow(color_1.getBlue() - color_2.getBlue(), 2);
47         }
48       }
49
```

```
50          sum /= 3;
51          sum /= h;
52          sum /= w;
53
54          if (min > sum) {
55            min = sum;
56          }
57
58          //MSE値を出力
59          System.out.println("MSE");
60          System.out.println(sum);
61
62          //PSNR値を計算
63          double a = Math.sqrt(sum);
64          double positiveValue1000 = max_2 / a;
65
66          //PSNR値を出力
67          System.out.println("PSNR");
68          System.out.println(20 * Math.log(positiveValue1000));
69
70        } catch (IOException e) {
71          e.printStackTrace();
72        }
73      }
74  }
```

[Random_RScode.mw]

```
restart;

ts := time();

#ガロア体GF(2^8)の定義
G8 := GF(2, 8, alpha^8+alpha^4+alpha^3+alpha^2+1);

a := G8:-ConvertIn(alpha);

#QRコード、バージョン1、誤り訂正レベルMの定義（https://sites.google.com/a/osshc.co.cc/web/studies/it/qr)
#埋め草シンボルの数khatの定義

n := 26; k := 16; khat := 9;

#誤り訂正コード語数10の生成多項式GPの定義（a^t の指数 t のみを並べたベクトルで表現、https://sites.google.com/a/osshc.co.cc/web/studies/it/qr)

GP := Vector(n-k+1, [0, 251, 67, 46, 61, 118, 70, 64, 94, 32, 45]);

#情報シンボルFの定義（FP は a^t の 指数 t のみを並べたベクトル（t=-1 のときはシンボルが0を表す）、https://sites.google.com/a/osshc.co.cc/web/studies/it/qr)

#FUKUDAのコード語[5, 194, 45, 10, 152, 16, conv[0], 49, 252, 29, 217, 113, 168, 24, 196, 216]
FP := Vector(k, [5, 194, 45, 10, 152, 16, -1, 122, 100, 122, 100, 122, 100, 122, 100, 122]);

F := Vector(k);

for i to k do
  if FP[i] >= 0 then
    F[i] := G8:-`^`(a, FP[i])
  else
    F[i] := G8:-input(0)
  end if
end do;

#ハミング距離の比較対象

Mican := [10001100, 10101101, 110000, 10011011, 11111, 11111100, 10001111, 11001000, 11000011, 11, 11000000, 11000011, 10100011, 11111011, 101111, 10001100, 10110101, 11000110, 110011];

#min_distanceの初期化
min_distance := 8*n;

#ここから繰り返す
for REPEAT to 1 do

```

```
#生成行列Gの作成
G := Matrix(k, n);

for i to k do
  for j to n do
    G[i, j] := G8:-input(0)
  end do
end do;

for i to k do
  for j to 11 do
    G[i, i-1+j] := G8:-`^`(a, GP[j])
  end do
end do;

#符号に現れる、情報シンボルの位置(1～(k-khat))と埋め草シンボルの位置(k-
    khat～nのどこかのkhat個)を, deltaで定義
delta := Vector(k);

for i to k-khat do
  delta[i] := i
end do;

myrand := rand(k-khat+1 .. n);
myset := {};
while nops(myset) <> khat do
  myset := myset union {myrand()}
end do;

mylist := convert(myset, list);
for i to khat do
  delta[i+k-khat] := mylist[i]
end do;

#掃き出し法によりGをランダム法で使う(I|P)に変形
#Iの後半の列はPのどこかの列にランダムに入る

for i to k do
  inv := G8:-inverse(G[i, delta[i]]);
  for j to n do
    G[i, j] := G8:-`*`(inv, G[i, j])
  end do;
  for l to i-1 do
    tmp := G[l, delta[i]];
    for j to n do
      G[l, j] := G8:-`-`(G[l, j], G8:-`*`(tmp, G[i, j]))
    end do
  end do;
  for l from i+1 to k do
    tmp := G[l, delta[i]];
    for j to n do
```

```
94        G[l, j] := G8:-‘-‘(G[l, j], G8:-‘*‘(tmp, G[i, j]))
95      end do
96    end do
97  end do;
98
99  #符号の計算 C=F*G
100 C := Vector(n);
101
102 for i to n do
103   C[i] := G8:-input(0);
104   for j to k do
105     C[i] := G8:-‘+‘(C[i], G8:-‘*‘(F[j], G[j, i]))
106   end do
107 end do;
108
109 #符号 C(Rx) を表示してみる
110 Rx := [seq(convert(G8:-output(C[i]), binary, decimal), i = 1 .. n)];
111
112 #ハミング距離の計算
113 distance := 0;
114
115 for i to nops(Mican) do
116   number_rx := Rx[nops(Rx)-nops(Mican)+i];
117   number_mi := Mican[i];
118   for j from 1 to 8 do
119     if number_rx mod 10 != number_mi mod 10 then
120       distance := add(distance, k = 1 .. 1)+1
121     end if;
122     number_rx := (add(number_rx, k = 1 .. 1)-add(number_rx mod 10), k = 1 .. 1))*(1/10);
123     number_mi := (add(number_mi, k = 1 .. 1)-add(number_mi mod 10, k = 1 .. 1))*(1/10)
124   end do
125 end do;
126
127 #最小ハミング距離を求める
128 if distance < min_distance then
129   min_distance := distance;
130   kekka := Rx
131 end if
132
133 #繰り返し終了
134 end do;
135
136 #計算時間の表示
137 print(time()-ts);
138
139 #最小ハミング距離の表示
140 print(min_distance);
141
142 #RS符号の表示
143 print(kekka)
```