Задача №3

polynom.rb

```ruby
require 'matrix'

class Polynom
  attr_reader :coefficients, :power, :truth_table

  def Polynom.truth_vector(*vector)
    coefficients = Polynom.get_system(Math.log2(vector.size).to_i)
    constants = Matrix.column_vector vector
    solutions = (coefficients.inverse * constants).map {|e| e.to_int % 2}
    new solutions.column 0
  end

  def Polynom.[](*coefficients)
    new coefficients
  end

  def value_for(vector)
    @truth_table[vector]
  end

  def truth_vector
    Vector.elements @truth_table.values
  end

  private
  def is_pow2?(n)
    n & (n - 1) == 0
  end

  def is_binary?(n)
    n == 1 || n == 0
  end

  def initialize(coefficients, values=nil)
    raise "Please specify 2^n coefficients" if !is_pow2? coefficients.size
    raise "Valid coefficients: 0 or 1" if coefficients.any? {|n| !is_binary? n}

    @power = Math.log2(coefficients.size).to_i
    @coefficients = Vector.elements coefficients
    values = values ? values : Polynom.get_truth_vector(coefficients, @power)
    @truth_table = Hash[*Polynom.arg_table(@power).zip(values).flatten(1)]
  end

```

```ruby
  def Polynom.get_combinations(variables, power=nil)
    power = power ? power : variables.size
    Array.new(power) do |key|
      variables.combination(key+1).to_a
    end.flatten(1)
  end

  def Polynom.multiply(variables, power=variables.size)
    combinations = Polynom.get_combinations(variables, power)
    [1] + combinations.map! do |combination|
      combination.reduce(:&)
    end
  end

  def Polynom.arg_table(vars)
    Array.new(2**vars) do |row|
      Array.new(vars) {|column| row[column]}
    end.sort {|a, b| a.count(1) <=> b.count(1)}
  end

  def Polynom.get_system(vars)
    Matrix.rows arg_table(vars).map {|args| multiply(args)}
  end

  def Polynom.get_truth_vector(coefficients, power)
    elements = arg_table(power).map do |values|
      mononoms = Vector.elements(Polynom.multiply values)
      values = mononoms.inner_product(coefficients) % 2
    end
    Vector.elements elements
  end
end
```

rm.rb

```ruby
require 'matrix'
require './polynom.rb'

class RM
  attr_reader :r, :m, :matrix

  def initialize(r, m)
    raise "r must be >= 0 and m must be >= r" if r < 0 or r > m
    @r, @m = r, m
    calculate_gen_matrix
  end

```

```
  def detects
    weight - 1
  end

  def corrects
    (weight - 1)/2
  end

  def information_rate
    Rational(dimension, length)
  end

  def notation
    [length, dimension, weight]
  end

  def to_s
    "RM(%d, %d)" % [@r, @m]
  end

  #n
  def length
    2**@m
  end
  alias :size :length

  #k
  def dimension
    matrix.row_size
  end

  # Minimum Hamming weight
  # d_min
  def weight
    2 ** (@m-@r)
  end

  def ==(other)
    other.r == @r && other.m == @m
  end

  def RM.detecting(errors, min_dimension)
    difference = Math.log2(errors + 1).ceil
    RM.by_parameters(difference, min_dimension)
  end

```

```
  def RM.correcting(errors, min_dimension)
    difference = Math.log2(Rational(1, 2) + errors).ceil + 1
    RM.by_parameters(difference, min_dimension)
  end

  private

  def RM.by_parameters(difference, min_dimension)
    r, m = 0, difference
    begin
      rm = RM.new(r, m)
      if m - r > difference
        r+=1
      else
        r = 0
        m += 1
      end
    end while rm.dimension < min_dimension
    rm
  end

  def calculate_gen_matrix
    arguments = Array.new(2**@m) do |row|
      Array.new(@m) { |column| row[@m - column - 1] }
    end
    columns = arguments.map {|row| Polynom.multiply row, @r}
    @matrix = Matrix.columns columns
  end
end
```

reed_coder.rb

```
module Math
  def Math.factorial(n)
    1.upto(n).inject(1) {|result, element| result * element}
  end
  def Math.choose(n, k)
    return 0 if k > n
    Rational(factorial(n), (factorial(k) * factorial(n-k)))
  end
end

module Statistics
  def Statistics.mode(array)
    array.group_by {|value| value}.values.max_by(&:size).first
  end
end
```

```

class ReedCoder
  attr_reader :code

  def initialize(code)
    @code = code
  end

  def encode_matrix(input)
    input.to_a.map {|vector| encode_vector vector}
  end

  def encode_vector(vector)
    result = Matrix.row_vector(vector) * code.matrix
    result.row(0).map {|element| element % 2}
  end

  def decode_vector(vector)
    result = []
    p 'start'
    # Main loop -- decreasing the order of the code
    code.r.downto(0) do |order|
      # The number of symbols of the information vector that we will be able to
      # calculate from this order.
      symbols = Math.choose(code.m, order).to_i

      # The number of bits of the vector per checksum for this order
      monomials = 2**order

      # The number of checksums per symbol for this order
      checksums = vector.size / monomials

      symbols.downto(1) do |symbol|
        # Offset of the first bit of the vector that is used in the sum
        offset = 0

        # Distance between the bits of the vector used in the sums
        distance = 2**(symbols - symbol)

        # The size of a block of checksums
        block_size = monomials * distance

        # The number of blocks of checksums for this symbol
        blocks = vector.size / block_size

        sums = []
```

```
        p "symbol %s, blocks %s, distance %s" % [symbol, blocks, distance]
        blocks.times do |block|
          distance.times do
            p sums.size
            sum = 0
            monomials.times do |monomial|

              sum += vector[offset + monomial*distance]
            end
            sums << sum
            offset += 1
          end
          offset = block * block_size
        end
        result << Statistics.mode(sums)
      end

      symbols.times do |symbol|

      end
      vector = adjust(vector, symbols, order)
    end
    p 'end'
    Vector.elements result.map {|element| element % 2}
  end

  def adjust(vector, coefficients, power)
    number = Math.choose(@code.m, power).to_i
    offset = 0.upto(power).inject(0) do |result, k|
      result+= Math.choose(@code.m, k)
    end.to_i

    vector = Vector.elements vector
    number.times do |index|
      vector = vector + code.matrix.row(offset - 1 - index) * coefficients[index]
    end
    vector.map {|element| element % 2}
  end

  def decode_matrix(matrix)
    matrix.to_a.map {|vector| decode_vector vector}
  end
end
```

Пораждаща матрица на $RM\,(1,4)$ (с параметри $[16,5,8]_2$):

$$\begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Нека $a_i = (a_0, \cdots, a_5)$ е информационен вектор на предадения вектор c_i. С помощта на несистематичния декодер на Рид ще определим информационните вектори на следните предадени вектори:

$$\begin{aligned} c_1 &= (1,0,0,0,0,1,0,1,1,1,1,0,0,1,0,1) \\ c_2 &= (1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1) \\ c_3 &= (1,1,1,0,1,1,1,0,1,1,1,0,1,1,1,0) \end{aligned}$$

$$\begin{array}{llll} a_{4_1} = 1+0 = 1 & a_{3_1} = 1+0 = 1 & a_{2_1} = 1+0 = 1 & a_{1_1} = 1+1 = 0 \\ a_{4_2} = 0+0 = 0 & a_{3_2} = 0+0 = 0 & a_{2_2} = 0+1 = 1 & a_{1_2} = 0+1 = 1 \\ a_{4_3} = 0+1 = 1 & a_{3_3} = 0+0 = 0 & a_{2_3} = 0+0 = 0 & a_{1_3} = 0+1 = 1 \\ a_{4_4} = 0+1 = 1 & a_{3_4} = 1+1 = 0 & a_{2_4} = 0+1 = 1 & a_{1_4} = 0+0 = 0 \\ a_{4_5} = 1+1 = 0 & a_{3_5} = 1+1 = 0 & a_{2_5} = 1+0 = 1 & a_{1_5} = 0+0 = 0 \\ a_{4_6} = 1+0 = 1 & a_{3_6} = 1+0 = 1 & a_{2_6} = 1+1 = 0 & a_{1_6} = 1+1 = 0 \\ a_{4_7} = 0+1 = 1 & a_{3_7} = 0+0 = 0 & a_{2_7} = 1+0 = 1 & a_{1_7} = 0+0 = 0 \\ a_{4_8} = 0+1 = 1 & a_{3_8} = 1+1 = 0 & a_{2_8} = 0+1 = 1 & a_{1_8} = 1+1 = 0 \end{array}$$

На базата на мажоритарната логика, можем да заключим, че информационните символи на изходната дума са както следва: $a_1 = 0, a_2 = 1, a_3 = 0, a_4 = 1$. Модифицираме c_1 с получените досега данни:

$$\begin{aligned} c_{1'} = c_1 + a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4 = c_1 + v_2 + v_4 = \\ = (1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1) \end{aligned}$$

Отново с оглед на това, че най-често срещаната стойност във вектора $c_{1'}$ е 1, заключваме, че $a_0 = 1$, имаме грешки в третия и десетия бит и окончателния вид на информационния вектор е:

$$a_1 = (1,0,1,0,1)$$

Чрез изчисления, аналогични на горните, определяме и информационните вектори на c_2 (получен без грешки) и c_3 (получен с грешки в четвърти, осми и дванайсти бит):

$$\begin{aligned} a_2 &= (1,1,1,1,1) \\ a_3 &= (1,0,0,0,0) \end{aligned}$$