УНИВЕРСИТЕТ ИТМО

ФГАОУ ВО «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

ПРИКЛАДНАЯ МАТЕМАТИКА

Лабораторная работа №5

Циклическое кодирование

Лабушев Тимофей

Группа P3302

Санкт-Петербург

2019

Цель работы

Получить практические навыки построения циклического кода по заданным характеристикам и проверка его свойства по обнаружению и исправлению ошибок.

Задание

1. Вычислить параметры кода и найти образующий многочлен, воспользовавшись таблицей неприводимых многочленов.
2. Проверить, имеются ли ошибки в исследуемой комбинации, при наличии ошибок – исправить их.
3. Провести программный контроль выполнения на примере случайных кодовых комбинаций.

Исходный код

Кодирование

```
#lang racket

(require data/bit-vector math/base "bit-utils.rkt")

(provide encode-cc decode-cc)

; https://www.partow.net/programming/polynomials/index.html
(define (generator-polynomial bits)
  (string->bit-vector (match bits
    [2 "111"]
    [3 "1011"]
    [4 "10011"]
    [5 "100101"]
    [6 "1000011"]
    [7 "10000011"]
    [8 "100011101"]
)))

(define (encode-cc data)
  (define info-bits (bit-vector-length data))
  (define check-bits (exact-round
    (log2 (+ (add1 info-bits) (log2 (add1 info-bits))))))
  (define polynomial (generator-polynomial check-bits))

  (define message (pad-trailing-bit-vector data (+ info-bits check-bits)))
  (define rem (modulo2-rem message polynomial))
  (or-bit-vectors message rem))

(define (decode-cc message)
  (define message-bits (bit-vector-length message))
  (define check-bits (exact-round (log (add1 message-bits) 2)))
  (define info-bits (- message-bits check-bits))
```

```
  (define polynomial (generator-polynomial check-bits))
  (define rem (modulo2-rem message polynomial))

  (cond
    [(= 0 (bit-vector-popcount rem)) (bit-vector-copy message 0 info-bits)]
    [else
      (define bit-i (lookup-error-bit message polynomial))
      (for/bit-vector #:length info-bits ([(b i) (in-indexed (in-bit-vector message))])
        (if (= i bit-i) (not b) b))]))

(define (lookup-error-bit message polynomial)
  (define msg-len (bit-vector-length message))
  (define table (for/hash ([error-bit-i (in-range msg-len)])
    (define error-msg (for/bit-vector ([i (in-range msg-len)]) (= i error-bit-i)))
    (define chksum (modulo2-rem error-msg polynomial))
    (values chksum error-bit-i)))
  (hash-ref table (modulo2-rem message polynomial)))

(define (log2 x) (log x 2))
```

Битовые операции

```
#lang racket

(require data/bit-vector)

(provide or-bit-vectors xor-bit-vectors
         pad-leading-bit-vector pad-trailing-bit-vector
         shl-bit-vector shr-bit-vector modulo2-rem)

(define (or-bit-vectors av bv) (zip-bit-vectors av bv or))
(define (xor-bit-vectors av bv) (zip-bit-vectors av bv xor))

(define-syntax-rule (zip-bit-vectors av bv op)
  (for/bit-vector ([a (in-bit-vector av)] [b (in-bit-vector bv)]) (op a b)))

(define (shl-bit-vector v) (shift-bit-vector-indexes v add1))
(define (shr-bit-vector v) (shift-bit-vector-indexes v sub1))

(define (pad-leading-bit-vector src-vec new-size [pad #f])
  (define pad-len (- new-size (bit-vector-length src-vec)))
  (for/bit-vector ([i (in-range new-size)])
    (if (< i pad-len) pad (bit-vector-ref src-vec (- i pad-len)))))

(define (pad-trailing-bit-vector src-vec new-size [fill #f])
  (for/bit-vector #:length new-size #:fill fill
    ([x (in-bit-vector src-vec)]) x))

(define (shift-bit-vector-indexes v shifter)
  (define len (bit-vector-length v))
  (for/bit-vector ([i (in-range len)])
    (bit-vector-ref v (wrap-index (shifter i) len))))

(define (wrap-index i len) (cond
  [(< i 0) (sub1 len)]
  [(= i len) 0]
```

```
  [else i]))

(define (modulo2-rem divident divisor)
  (define result (bit-vector-copy divident))
  (define (modulo2-rem-loop bit-i)
    (for ([(d j) (in-indexed (in-bit-vector divisor))])
      (define res-bit (bit-vector-ref result (+ bit-i j)))
      (bit-vector-set! result (+ bit-i j) (xor res-bit d)))
    (define (skip-zeroes i)
      (cond
        [(> i (- (bit-vector-length result) (bit-vector-length divisor))) result]
        [(not (bit-vector-ref result i)) (skip-zeroes (add1 i))]
        [else (modulo2-rem-loop i)]))
    (skip-zeroes bit-i))
  (modulo2-rem-loop 0))
```

Тестирование

```
#lang racket

(require rackunit rackunit/text-ui data/bit-vector "coder.rkt")

(define coder-tests
  (test-suite "coder.rkt tests"
    (test-case "encode-decode"
      (define cases '("10011" "1100001010" "1100"))
      (for ([data (in-list cases)])
        (define src (string->bit-vector data))
        (define encoded (encode-cc src))
        (define decoded (decode-cc encoded))
        (check-equal? (bit-vector->string decoded) data)))

    (test-case "single bit error"
      (define src (string->bit-vector "1100"))
      (define encoded (encode-cc src))
      (for ([i (in-range (bit-vector-length encoded))])
        (define error-msg (bit-vector-copy encoded))
        (bit-vector-set! error-msg i (not (bit-vector-ref error-msg i)))
        (define decoded (decode-cc error-msg))
        (check-equal? (bit-vector->string decoded) "1100")))))

(run-tests coder-tests)
```

Выводы

В ходе выполнения лабораторной работы были изучены прицины построения и декодирования циклических кодов.