УНИВЕРСИТЕТ ИТМО

ФГАОУ ВО «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Факультет программной инженерии и компьютерной техники

ПРИКЛАДНАЯ МАТЕМАТИКА

Лабораторная работа №3

Арифметическое кодирование

Лабушев Тимофей

Группа P3302

Санкт-Петербург

2019

Цель работы

Получить практические навыки кодирования и декодирования текстового файла арифметическим кодом.

Задание

1. Реализовать процедуру построения арифметического кода
2. Предусмотреть ввод последовательности символов для кодирования
3. Вычислить коэффициент сжатия данных как процентное отношение длины закодированного файла к длине исходного файла
4. Декодировать полученную последовательность, сравнить полученный файл с исходным текстом

Исходный код

Кодирование

```
#lang typed/racket

(provide make-code-table encode decode)

(define-type Char-Freq-Hash (HashTable Char Nonnegative-Integer))
(define-type Char-Segment-Hash (HashTable Char Segment-Bounds))
(define-type Code-Point Nonnegative-Exact-Rational)
(define-type Segment-Bounds (Pair Code-Point Code-Point))

(: char-freqs (-> (Listof Char) Char-Freq-Hash))
(define (char-freqs charlst)
  (: folder (-> Char Char-Freq-Hash Char-Freq-Hash))
  (define (folder char freq-hash) (hash-update freq-hash char add1 (λ () 0)))
  (foldl folder (ann (make-immutable-hash) Char-Freq-Hash) charlst))

(: char-segments (-> Char-Freq-Hash Char-Segment-Hash))
(define (char-segments freq-hash)
  (define total-char-cnt (apply + (hash-values freq-hash)))
  (define bound : Code-Point 0)
  (for/hash : Char-Segment-Hash ([(char cnt) (in-hash freq-hash)])
    (define lower-bound bound)
    (set! bound (+ bound (/ cnt total-char-cnt)))
    (values char (cons lower-bound bound))))

(: make-code-table (-> String Char-Segment-Hash))
(define make-code-table (compose (compose char-segments char-freqs) string->list))

(: encode (-> (Sequenceof Char) Char-Segment-Hash Code-Point))
(define (encode input segments)
  (car (for/fold ([bounds : Segment-Bounds (cons 0 1)]) ([char input])
    (match-define (cons lower upper) bounds)
```

```
    (match-define (cons lbound ubound) (hash-ref segments char))
    ; Assert nonnegativity for the type checker (noop in practice since upper > lower)
    (define bounds-diff (abs (- upper lower)))
    (cons (+ lower (* lbound bounds-diff)) (+ lower (* ubound bounds-diff))))))

(: decode (-> Code-Point Index Char-Segment-Hash String))
(define decode (λ (input text-len segments)
  (define segment-list (hash->list segments))
  (car (for/fold ([acc : (Pair String Code-Point) (cons "" input)]) ([_ (in-range text-len)])
    (match-define (cons decoded bound) acc)
    (match-define-values (char lbound ubound) (decode-lookup-char bound segment-list))
    (cons (~a decoded char) (abs (/ (- bound lbound) (- ubound lbound))))))))

(: decode-lookup-char
   (-> Code-Point (Listof (Pair Char Segment-Bounds)) (Values Char Code-Point Code-Point)))
(define (decode-lookup-char bound segment-list)
  (match-define (cons (cons char (cons lbound ubound)) tail) segment-list)
  (if (and (<= lbound bound) (< bound ubound))
    (values char lbound ubound)
    (decode-lookup-char bound tail)))
```

Ввод

```
#lang racket

(require math/bigfloat "coder.rkt")

(define (charlist-mismatch-index lst1 lst2 [index 0])
  (cond
    [(and (not (empty? lst1)) (equal? (car lst1) (car lst2)))
     (charlist-mismatch-index (cdr lst1) (cdr lst2) (add1 index))]
    [else index]))

(define (do-code input bits)
  (bf-precision bits)

  (define codetable (make-code-table input))
  (define encoded-precise (encode input codetable))
  (define encoded-trunc (bf encoded-precise))
  (define decoded (decode (bigfloat->rational encoded-trunc) (string-length input) codetable))

  (define precision (charlist-mismatch-index (string->list input) (string->list decoded)))
  (define input-bits (* 8 (bytes-length (string->bytes/utf-8 input))))
  (define compression-ratio (exact->inexact (* 100 (/ bits input-bits))))

  (display (~a "Encoded: " (bigfloat->string encoded-trunc) "\n"))
  (display (~a "Decoded: " decoded "\n"))
  (display (~a "Encoding precision: " precision "\n"))
  (display (~a "Compression ratio: " compression-ratio "%\n")))

(define (main)
  (display "Enter the text to encode:\n")
  (define input (read-line))
  (display "Enter desired precision (in bits, 53 for double precision)\n")
  (define bits (read))
  (cond
    [(not (integer? bits)) (display "Expected a number\n")]
```

```
    [else (do-code input bits)]))

(main)
```

Пример работы программы

```
Enter the text to encode:
Fly me to the moon, let me play among the stars, and let me see what spring is like on Jupyter and Mars
Enter desired precision (in bits, 53 for double precision)
450
Encoded: 0.98217839821460216142080590435137654177818958962232883053288114104860106024291031466569477120825557080953424165783775107
Decoded: Fly me to the moon, let me play among the stars, and let me see what spring is like on Jupyter and Mars
Encoding precision: 103
Compression ratio: 54.61165048543689%
```

Выводы

В ходе выполнения лабораторной работы было установлено, что для кодирования сообщений достаточной длины арифметическим кодированием необходима высокая точность представления чисел с плавающей запятой.