МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа №5**

по дисциплине: Теория информации

тема: «Арифметическое кодирование»

Выполнил: ст. группы ПВ-233

Ситников Алексей Павлович

Проверил: Твердохлеб Виталий

Викторович

Белгород 2025 г.

Задания лабораторной работы

1. Построить обработчик, реализующий функцию арифметического кодирования.

2. В качестве исходных данных, подлежащих обработке, использовать последовательности из работы №2. Для полученных результатов рассчитать показатели сжатия. Сравнить с полученными в работе №2.

1. Построить обработчик, реализующий функцию арифметического кодирования.

package main

import (

    "fmt"

    "math/big"

    "reflect"

    "strings"

)

// Кодирует сообщение, возвращает закодированную дробь и таблицу декодирования

func codeMessage(message string, x float64) (\*big.Rat, map[rune][2]\*big.Rat) {

    message += "\x00"

    letters := make(map[rune]int)

    decodeTable := make(map[rune][2]\*big.Rat)

    // Считаем частоту символов

    for \_, char := range message {

        letters[char]++

    }

    var oldFraction [2]\*big.Rat

    oldFraction[0] = big.NewRat(0, 1)

    oldFraction[1] = big.NewRat(0, 1)

    // Создаем таблицу декодирования

    for char, count := range letters {

        frac := big.NewRat(int64(count), int64(len(message)))

        decodeTable[char] = [2]\*big.Rat{oldFraction[1], new(big.Rat).Add(oldFraction[1], frac)}

        oldFraction = decodeTable[char]

    }

    // Начальная дробь для кодирования

    var newFraction [2]\*big.Rat

    newFraction[0] = big.NewRat(0, 1)

    newFraction[1] = big.NewRat(0, 1)

    // Перемножаем дроби для кодирования

    for \_, char := range message {

        t := decodeTable[char]

        if newFraction[0].Cmp(big.NewRat(0, 1)) == 0 {

            newFraction = t

        } else {

            left := new(big.Rat).Set(newFraction[0])

            right := new(big.Rat).Set(newFraction[1])

            newFraction[0] = new(big.Rat).Add(left, new(big.Rat).Mul(new(big.Rat).Sub(right, left), t[0]))

            newFraction[1] = new(big.Rat).Add(left, new(big.Rat).Mul(new(big.Rat).Sub(right, left), t[1]))

        }

    }

    // Возвращаем закодированную дробь

    return new(big.Rat).Add(newFraction[0], newFraction[1]), decodeTable

}

// Декодирует закодированное сообщение с помощью таблицы декодирования

func decodeMessage(codedMsg \*big.Rat, table map[rune][2]\*big.Rat) string {

    var message strings.Builder

    for {

        var tup [2]\*big.Rat

        var ch rune

        for c, t := range table {

            if codedMsg.Cmp(t[0]) >= 0 && codedMsg.Cmp(t[1]) < 0 {

                tup = t

                ch = c

                break

            }

        }

        if ch == '\x00' {

            break

        }

        message.WriteRune(ch)

        codedMsg.Sub(codedMsg, tup[0])

        codedMsg.Quo(codedMsg, new(big.Rat).Sub(tup[1], tup[0]))

    }

    return message.String()

}

func printData(message string) {

    undef := true

    prev := 1.0

    curr := 1.0

    var code \*big.Rat

    var table map[rune][2]\*big.Rat

    for undef || decodeMessage(code, table) == message {

        code, table = codeMessage(message, curr)

        if decodeMessage(code, table) != message {

            break

        }

        prev = curr

        curr \*= 1.1

        undef = false

    }

    code, table = codeMessage(message, prev)

    fmt.Printf("Дробь: %s\n\n", code.RatString())

    fmt.Println("Таблица декодирования:")

    for key, val := range table {

        fmt.Printf("'%c' => [%s, %s]\n", key, val[0].RatString(), val[1].RatString())

    }

    fmt.Println("\nДекодированное сообщение:")

    fmt.Println(decodeMessage(code, table))

    // Коэффициент сжатия (исправленная версия)

    num := code.Num()

    den := code.Denom()

    codeSize := int64(reflect.TypeOf(num).Size() + reflect.TypeOf(den).Size())

    messageSize := int64(len(message))

    compressionRatio := float64(messageSize) / float64(codeSize)

    fmt.Printf("\nКоэффициент сжатия: %.2f\n", compressionRatio)

}

func main() {

    s1 := "в чащах юга жил бы цитрус? да но фальшивый экземпляр!"

    s2 := "Victoria nulla est, Quam quae confessos animo quoque subjugat hostes"

    printData(s1)

    fmt.Println("--------------------------------------------")

    printData(s2)

}

Вывод программы:

Изображение выглядит как снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

2. В качестве исходных данных, подлежащих обработке, использовать последовательности из работы №2. Для полученных результатов рассчитать показатели сжатия. Сравнить с полученными в работе №2.

Для сообщения “в чащах юга жил бы цитрус? да но фальшивый экземпляр!” коэффициент сжатия при арифметическом кодировании K = 1.325, а при кодировании кодом Хаффмана K = 1.27.

Для сообщения “Victoria nulla est, Quam quae confessos animo quoque subjugat hostes” коэффициент сжатия при арифметическом кодировании K = 1.545, а при кодировании кодом Хаффмана K = 1.22.

**Вывод:** арифметическое кодирование показало более высокую степень сжатия в отличие от алгоритма Хаффмана путём устранения структурной избыточности сообщения.