

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"
)

// Кодирование сообщения, 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 чашах 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:

```
Пробь: 2760804244648205118055541897511669758846576977070059145772892362290945918912459135902349635069/6103184471212967314410237482608474455733439701717895171538080866085
107734526501105239629430784
```

Таблица декодирования:

```
'а' => [7/32, 23/96]
'!' => [7/16, 43/96]
'д' => [11/24, 15/32]
'ц' => [0, 1/96]
'й' => [1/48, 1/32]
'а' => [1/24, 3/32]
'ч' => [13/96, 7/48]
'и' => [1/4, 3/32]
'ы' => [9/32, 29/96]
'т' => [29/96, 5/16]
'к' => [13/24, 53/96]
'ж' => [53/96, 9/16]
'о' => [1/96, 1/48]
'е' => [1/32, 1/24]
'б' => [3/32, 5/48]
'р' => [17/96, 19/96]
'х' => [23/96, 1/4]
'ш' => [15/32, 23/48]
'ф' => [49/96, 25/48]
' ' => [11/32, 7/16]
'м' => [43/96, 11/24]
'э' => [25/48, 17/32]
'г' => [1/8, 13/96]
'с' => [31/96, 1/3]
'ю' => [23/48, 47/96]
'я' => [47/96, 1/2]
'н' => [1/2, 49/96]
'щ' => [17/32, 13/24]
'п' => [11/96, 1/8]
'? ' => [19/96, 5/24]
'' => [5/24, 7/32]
'э' => [5/48, 11/96]
'у' => [5/16, 31/96]
'ь' => [1/3, 11/32]
'л' => [7/48, 17/96]
```

Декодированное сообщение:

и

Коэффициент сжатия: 5.94

```
Пробь: 2909222055428845444200598337046086355928165157663903319156655433331707859482230613400677942036098406133215516/1874382957257121994007636357909965236331398542610802
708258051123510375121417574935090739347953935839040738687
```

Таблица декодирования:

```
'е' => [1/3, 28/69]
'i' => [12/23, 13/23]
'с' => [13/23, 41/69]
'v' => [53/69, 18/23]
'q' => [0, 1/23]
'' => [5/69, 2/23]
'o' => [2/23, 4/23]
'r' => [13/69, 14/69]
'h' => [58/69, 59/69]
'a' => [44/69, 50/69]
'g' => [68/69, 1]
'f' => [1/23, 4/69]
'u' => [14/69, 7/23]
's' => [28/69, 35/69]
'Q' => [35/69, 12/23]
'n' => [50/69, 53/69]
'j' => [4/69, 5/69]
',' => [4/23, 13/69]
'm' => [41/69, 43/69]
'b' => [43/69, 44/69]
'l' => [7/23, 1/3]
't' => [18/23, 58/69]
' ' => [59/69, 68/69]
```

Декодированное сообщение:

Коэффициент сжатия: 4.25

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.