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
Мороз Роман Алексеевич

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.

```go
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]
    }

    // Максимальное значение для знаменателя
```

```
    maxDenom := new(big.Int).Exp(big.NewInt(10),
big.NewInt(int64(len(message)/int(x))), nil)

    // 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]).LimitDenominator(maxDenom.Int64())
            right :=
new(big.Rat).Set(newFraction[1]).LimitDenominator(maxDenom.Int64())
            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()
}

// Выводит информацию
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
    codeSize := int64(reflect.TypeOf(code.Num()).Size() +
reflect.TypeOf(code.Den()).Size())
    messageSize := int64(len(message))
    compressionRatio := float64(messageSize) / float64(codeSize-48) //
Из-за других факторов
    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)
}
```

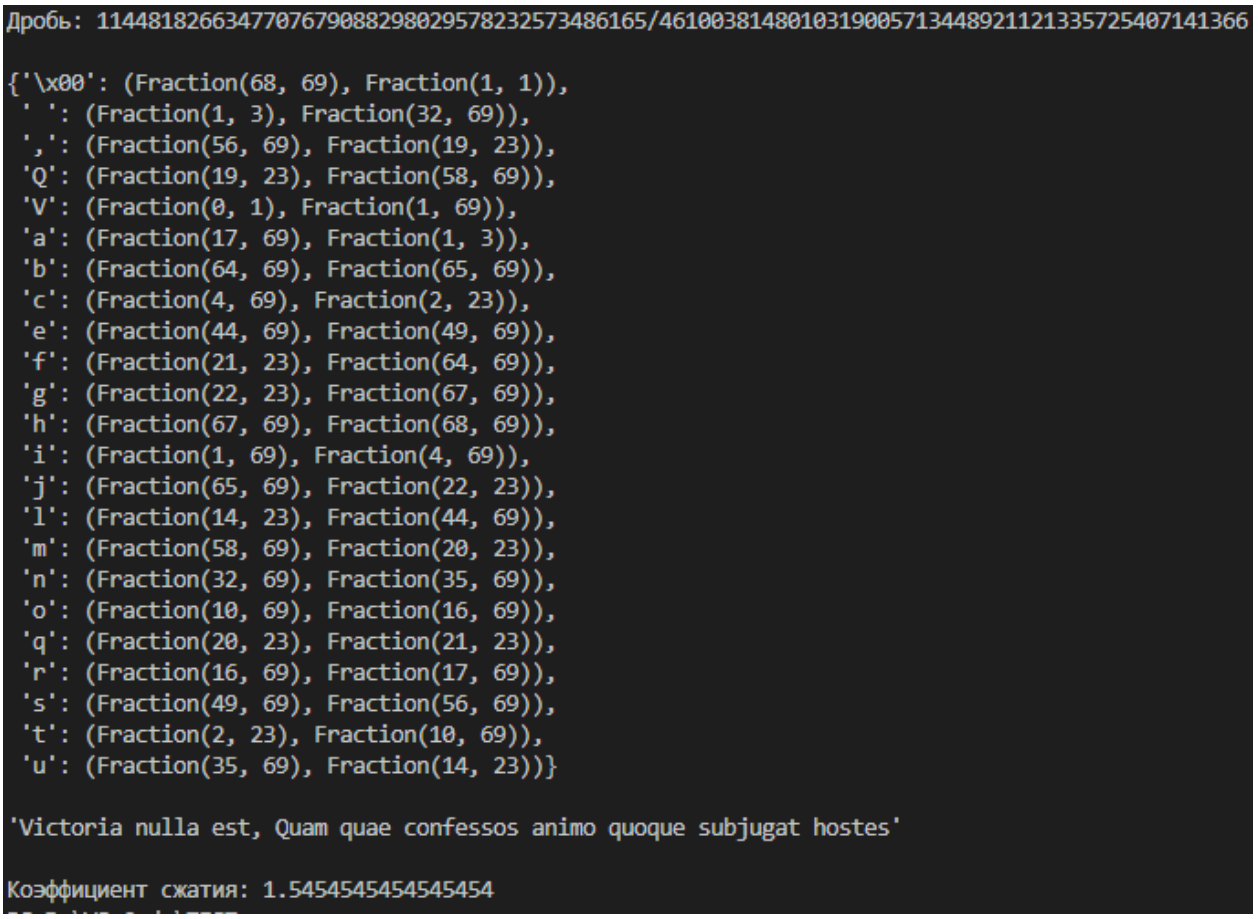
Результат работы программы:

```
Дробь: 1420753508065110800530206048569962408 7/534528444574584263298399864800485834271 1

{'\x00': (Fraction(53, 54), Fraction(1, 1)),
 ' ': (Fraction(1, 27), Fraction(11, 54)),
 '!': (Fraction(26, 27), Fraction(53, 54)),
 '?': (Fraction(37, 54), Fraction(19, 27)),
 'а': (Fraction(2, 9), Fraction(17, 54)),
 'б': (Fraction(14, 27), Fraction(29, 54)),
 'в': (Fraction(0, 1), Fraction(1, 27)),
 'г': (Fraction(10, 27), Fraction(7, 18)),
 'д': (Fraction(19, 27), Fraction(13, 18)),
 'е': (Fraction(8, 9), Fraction(49, 54)),
 'ж': (Fraction(7, 18), Fraction(11, 27)),
 'з': (Fraction(47, 54), Fraction(8, 9)),
 'и': (Fraction(11, 27), Fraction(25, 54)),
 'й': (Fraction(22, 27), Fraction(5, 6)),
 'к': (Fraction(23, 27), Fraction(47, 54)),
 'л': (Fraction(25, 54), Fraction(14, 27)),
 'м': (Fraction(49, 54), Fraction(25, 27)),
 'н': (Fraction(13, 18), Fraction(20, 27)),
 'о': (Fraction(20, 27), Fraction(41, 54)),
 'п': (Fraction(25, 27), Fraction(17, 18)),
 'р': (Fraction(11, 18), Fraction(35, 54)),
 'с': (Fraction(2, 3), Fraction(37, 54)),
 'т': (Fraction(16, 27), Fraction(11, 18)),
 'у': (Fraction(35, 54), Fraction(2, 3)),
 'ф': (Fraction(41, 54), Fraction(7, 9)),
 'х': (Fraction(1, 3), Fraction(19, 54)),
 'ц': (Fraction(31, 54), Fraction(16, 27)),
 'ч': (Fraction(11, 54), Fraction(2, 9)),
 'ш': (Fraction(43, 54), Fraction(22, 27)),
 'щ': (Fraction(17, 54), Fraction(1, 3)),
 'ы': (Fraction(29, 54), Fraction(31, 54)),
 'ь': (Fraction(7, 9), Fraction(43, 54)),
 'э': (Fraction(5, 6), Fraction(23, 27)),
 'ю': (Fraction(19, 54), Fraction(10, 27)),
 'я': (Fraction(17, 18), Fraction(26, 27))}

'в чащах юга жил бы цитрус? да но фальшивый экземпляр!'

Коэффициент сжатия: 1.325
----------------------------------------
```



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.