БУ ВО Ханты – Мансийского автономного округа – Югры

«Сургутский государственный университет»

Политехнический институт

Кафедра автоматики и компьютерный систем

**ЛАБОРАТОРНАЯ РАБОТА №3**

по дисциплине: «Основы защиты информации»

по теме: «Хафман»

Выполнила: студентка группы №609-11

Пашина Илона Руслановна

Приняла: старший преподаватель кафедры АиКС

Кривицкая Марина Александровна

Сургут

2025 г.

**Тема**: сжатие данных.

**Задание на лабораторную работу:**

1. Программно реализовать упаковку/распаковку по Хафману

**Среда реализации**: любая.

**Листинг**

package main

import (

"encoding/binary"

"fmt"

"io"

"math/big"

"os"

"sort"

)

type LetterCount struct {

Letter rune

Count int

Left \*LetterCount // 1

Right \*LetterCount // 0

Code string

}

func countLetters(text string) map[rune]int {

letterCounts := make(map[rune]int)

for \_, char := range text {

letterCounts[char]++

}

return letterCounts

}

func mapToArr(charCount map[rune]int) []\*LetterCount {

letterCounts := make([]\*LetterCount, 0, len(charCount))

for letter, count := range charCount {

letterCounts = append(letterCounts, &LetterCount{Letter: letter, Count: count})

}

return letterCounts

}

type ByCount []\*LetterCount

func (a ByCount) Len() int { return len(a) }

func (a ByCount) Swap(i, j int) { a[i], a[j] = a[j], a[i] }

func (a ByCount) Less(i, j int) bool { return a[i].Count < a[j].Count }

func buildHuffmanTree(counts []\*LetterCount) \*LetterCount {

if len(counts) == 0 {

return nil

}

sort.Sort(ByCount(counts))

for len(counts) > 1 {

left := counts[0]

right := counts[1]

combined := &LetterCount{

Count: left.Count + right.Count,

Left: left,

Right: right,

}

counts = counts[2:]

counts = append(counts, combined)

sort.Sort(ByCount(counts))

}

return counts[0]

}

func generateCodes(node \*LetterCount, code string) {

if node == nil {

return

}

if node.Left == nil && node.Right == nil {

node.Code = code

return

}

generateCodes(node.Left, "1"+code)

generateCodes(node.Right, "0"+code)

}

func codemessage(arrletter []\*LetterCount, message string) string {

cm := ""

for \_, val := range message {

for \_, letter := range arrletter {

if val == letter.Letter {

cm = letter.Code + cm

break

}

}

}

return cm

}

func saveTreeToFile(root \*LetterCount, filename string) error {

file, err := os.Create(filename)

if err != nil {

return fmt.Errorf("ошибка создания файла: %w", err)

}

defer file.Close()

if err := saveNode(file, root); err != nil {

return fmt.Errorf("ошибка записи дерева: %w", err)

}

return nil

}

func saveNode(w io.Writer, node \*LetterCount) error {

if node == nil {

if err := binary.Write(w, binary.LittleEndian, int8(0)); err != nil {

return fmt.Errorf("ошибка записи nil флага: %w", err)

}

return nil

}

if err := binary.Write(w, binary.LittleEndian, int8(1)); err != nil {

return fmt.Errorf("ошибка записи not nil флага: %w", err)

}

if err := binary.Write(w, binary.LittleEndian, node.Letter); err != nil {

return fmt.Errorf("ошибка записи Letter: %w", err)

}

if err := binary.Write(w, binary.LittleEndian, int32(node.Count)); err != nil {

return fmt.Errorf("ошибка записи Count: %w", err)

}

if err := saveNode(w, node.Left); err != nil {

return err

}

if err := saveNode(w, node.Right); err != nil {

return err

}

return nil

}

func readTreeFromFile(filename string) (\*LetterCount, error) {

file, err := os.Open(filename)

if err != nil {

return nil, fmt.Errorf("ошибка открытия файла: %w", err)

}

defer file.Close()

root, err := readNode(file)

if err != nil {

return nil, fmt.Errorf("ошибка чтения дерева: %w", err)

}

return root, nil

}

func readNode(r io.Reader) (\*LetterCount, error) {

var flag int8

if err := binary.Read(r, binary.LittleEndian, &flag); err != nil {

if err == io.EOF {

return nil, nil

}

return nil, fmt.Errorf("ошибка чтения флага nil: %w", err)

}

if flag == 0 {

return nil, nil

}

node := &LetterCount{}

if err := binary.Read(r, binary.LittleEndian, &node.Letter); err != nil {

return nil, fmt.Errorf("ошибка чтения Letter: %w", err)

}

var count int32

if err := binary.Read(r, binary.LittleEndian, &count); err != nil {

return nil, fmt.Errorf("ошибка чтения Count: %w", err)

}

node.Count = int(count)

left, err := readNode(r)

if err != nil {

return nil, err

}

node.Left = left

right, err := readNode(r)

if err != nil {

return nil, err

}

node.Right = right

return node, nil

}

func stringToBinaryFileBigInt(binaryString string, filename string) error {

num := new(big.Int)

num, ok := num.SetString(binaryString, 2)

if !ok {

return fmt.Errorf("ошибка преобразования строки в большое число")

}

file, err := os.Create(filename)

if err != nil {

return fmt.Errorf("ошибка создания файла: %w", err)

}

defer file.Close()

// big.Int в []byte и запись в файл.

numBytes := num.Bytes()

length := uint32(len(numBytes))

if err := binary.Write(file, binary.LittleEndian, length); err != nil {

return fmt.Errorf("ошибка записи длины массива: %w", err)

}

if \_, err := file.Write(numBytes); err != nil {

return fmt.Errorf("ошибка записи байтов в файл: %w", err)

}

return nil

}

func readBigIntFromFile(filename string) (string, error) {

file, err := os.Open(filename)

if err != nil {

return "", fmt.Errorf("ошибка открытия файла: %w", err)

}

defer file.Close()

num := new(big.Int)

var length uint32

if err := binary.Read(file, binary.LittleEndian, &length); err != nil {

return "", fmt.Errorf("ошибка чтения длины массива: %w", err)

}

numBytes := make([]byte, length)

\_, err = io.ReadFull(file, numBytes)

if err != nil {

return "", fmt.Errorf("ошибка чтения байтов из файла: %w", err)

}

num.SetBytes(numBytes)

// big.Int в бинарную строку

binaryString := num.Text(2)

return binaryString, nil

}

func main() {

text := "Helllhdfdfsvbdkjnvs.hdbv"

fmt.Println("Кодируемый текст: ", text)

mapletterCounts := countLetters(text) // словарь

arrletterCounts := mapToArr(mapletterCounts) // массив

root := buildHuffmanTree(arrletterCounts)

generateCodes(root, "")

codedtext := codemessage(arrletterCounts, text)

fmt.Println()

filename := "tree.bin"

err := saveTreeToFile(root, filename)

if err != nil {

fmt.Printf("Ошибка сохранения дерева: %v\n", err)

return

}

fmt.Println("Дерево успешно сохранено в", filename)

filenamems := "big\_number.bin"

err = stringToBinaryFileBigInt(codedtext, filenamems)

if err != nil {

fmt.Printf("Ошибка: %v\n", err)

return

}

fmt.Printf("Большое число из строки успешно записано в файл %s.\n\n", filenamems)

////////////////////////////////////////////////////////////

root, err = readTreeFromFile(filename)

if err != nil {

fmt.Printf("Ошибка чтения дерева: %v\n", err)

return

}

fmt.Println("Дерево успешно прочитано из", filename)

codedtext, err = readBigIntFromFile(filenamems)

if err != nil {

fmt.Printf("Ошибка чтения из файла: %v\n", err)

return

}

fmt.Println("Большое число успешно прочитано из", filenamems)

////////////////////////////////////////////////////////////

uncodetext := uncodemessage(root, codedtext)

fmt.Println("\nДекодированный текст: ", uncodetext)

fmt.Println()

}

func uncodemessage(node \*LetterCount, message string) string {

ucm := ""

now\_node := node

for i := len(message) - 1; i >= 0; i-- {

if message[i] == '1' && now\_node.Left != nil {

now\_node = now\_node.Left

} else if message[i] == '0' && now\_node.Right != nil {

now\_node = now\_node.Right

} else {

ucm += string(now\_node.Letter)

now\_node = node

if i != 0 {

i++

}

}

}

if message[0] == '1' && now\_node.Left != nil {

now\_node = now\_node.Left

} else if message[0] == '0' && now\_node.Right != nil {

now\_node = now\_node.Right

}

ucm += string(now\_node.Letter)

return ucm

}

func printHuffmanTree(node \*LetterCount, indent string) {

if node == nil {

return

}

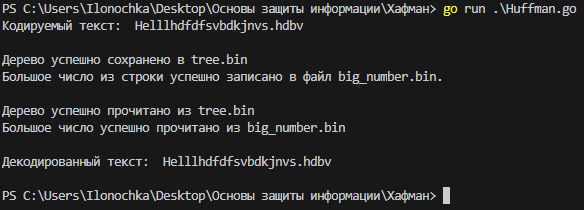
printHuffmanTree(node.Left, indent+" ")

fmt.Printf("%s%c (%d) Code: %s\n", indent, node.Letter, node.Count, node.Code)

printHuffmanTree(node.Right, indent+" ")

}

**Пример работы программы:**



**Вывод:**

В ходе выполнения лабораторной работы была написана программа по теме: сжатие Хафмана.