Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Защита информации и надёжность информационных систем»

**Лабораторная работа №8**

**Тема «СЖАТИЕ/РАСПАКОВКА ДАННЫХ НА ОСНОВЕ СТАТИСТИЧЕСКИХ МЕТОДОВ»**

Выполнил:

Студент 3 курса 7 группы ФИТ

Тимошенко Д. В.   
 Проверила:   
 асс. Николайчук А. Н.

Минск 2024

**Цель работы:** приобретение практических навыков использования статистических методов Шеннона − Фано и Хаффмана (Shannon-Fano and Huffman coding) для сжатия/распаковки данных.

**Задание на лабораторную работу**

1. Разработать авторское приложение в соответствии с целью лабораторной работы.

2. С помощью приложения выполнить прямое и обратное преобразования сообщения, состоящего из собственных имени и фамилии.

Можно использовать любой из известных методов сортировки символов массива. Метод кодировки (Шеннона − Фано, Хаффмана) использовать по указанию преподавателя.

При этом таблица отсортированных символов строится:

а) на основе данных, полученных в лабораторной работе № 2;

б) динамически, на основе анализа сжимаемого сообщения.

3. Определить эффективность (в сравнении с кодами ASCII) сжатия сообщения.

4. Результаты оформить в виде отчета по установленным правилам.

Каждый из естественных языков обладает избыточностью. Среди европейских языков белорусский и русский обладают одним из самых высоких уровней избыточности. Об этом можно судить по размерам русского перевода английского текста. Обычно он примерно на 20−30% больше.

Статистические алгоритмы позволяют создавать более короткие коды для часто встречающихся и более длинные – для редко встречающихся символов алфавита или конкретного сообщения. В первом случае метод считается статическим статистическим, во втором – динамическим статистическим: вероятностные свойства символов подсчитываются для конкретного сообщения или потока данных.

Частота или вероятность появления того или иного символа алфавита в произвольном сообщении, лежащая в основе алгоритмов, дали название этим алгоритмам и соответствующим методам.

Иногда эти методы называют также префиксными.

Методы относятся к классу «сжатие без потерь». Различие между двумя рассматриваемыми методами состоит лишь в особенностях формирования таблицы бинарных кодов.

Код Шеннона – Фано не является оптимальным (обеспечивает минимальную избыточность) в общем смысле, хотя и дает оптимальные результаты при некоторых распределениях вероятностей.

Для одного и того же распределения вероятностей можно построить, вообще говоря, несколько кодов Шеннона – Фано, и все они могут дать различные результаты.

Итак, необходимо выполнить следующие действия:

1) подсчитать вероятностные параметры символов алфавита А = {a i} (реализуется статическая версия алгоритма);

2) отсортировать – обычно в порядке убывания (невозрастания, т. е. могут иметь место повторяющиеся значения) вероятностей р(аi); р(аi) – вероятность появления в сжимаемом сообщении на произвольной позиции символа аi алфавита, т. е. создать таблицу символов алфавита, на основе которого генерируется сжимаемое сообщение;

3) каждому символу отсортированного множества поставить в соответствие бинарный код, для чего это множество (таблица) символов делится на две группы таким образом, чтобы каждая из групп имела приблизительно одинаковую суммарную частоту (вероятность).

• Метод основан на алгоритме оптимального префиксного кодирования алфавита: исходный алгоритм Хаффмана является оптимальным для посимвольного кодирования с известным входным распределением вероятностей, т. е. для отдельного кодирования несвязанных символов в таком потоке данных. Отличается от метода Шеннона – Фано лишь в части кодирования символов исходного алфавита.

В данном случае бинарные коды создаются на основе дерева, ветви которого обозначаются бинарными символами. Бинарным кодом символа исходного алфавита будет последовательность обозначений ветвей дерева от корня до листа, соответствующего этому символу. В основе бинарного кода лежит следующее положение.

Лемма. Для любого заданного алфавита (источника) с N > 2 символами существует оптимальный двоичный код, в котором два наименее вероятных символа (слова) имеют одну и ту же длину и отличаются лишь последним битом.

Построение дерева начинается с сортирования символов исходного алфавита в порядке убывания (невозрастания). Далее выбираются два символа (a i, a j) с наименьшими вероятностями (р(a i), р(a j)) и объединяются в узел. Ветви этого узла

обозначаются «1» и «0». Этот узел рассматривается далее как новый, виртуальный символ (a ij), которому будет соответствовать вероятность р(a ij) = р(a i) + р(a j). Такой виртуальный символ будет рассматриваться далее наравне с остальными символами исходного алфавита. Два его потомка из дальнейшего рассмотрения исключаются. Создаются новые узлы дерева по тому же принципу. Корень дерева образуют два символа с наибольшими вероятностями.

Приложение, для выполнения задания, написано на языке программирования JavaScript. Ниже представлены листинг функций, используемых во всех задания, листинг 1.1-1.3.

|  |
| --- |
| class TNode {  symbol: string;  frequency: number;  right: TNode | null;  left: TNode | null;  constructor(symbol: string, frequency: number) {  this.symbol = symbol;  this.frequency = frequency;  this.right = null;  this.left = null;  }  traverse(symbol: string, data: boolean[]): boolean[] | null {  if (this.right === null && this.left === null) {  return symbol === this.symbol ? data : null;  }  let left: boolean[] | null = null;  let right: boolean[] | null = null;  if (this.left !== null) {  left = this.left.traverse(symbol, [...data, false]);  }  if (this.right !== null) {  right = this.right.traverse(symbol, [...data, true]);  }  return left !== null ? left : right;  }  } |

Листинги 1.1 – Класс для построения узла дерева

|  |
| --- |
| class HuffmanTree {  nodes: TNode[] = [];  root: TNode | null = null;  frequencies: Map<string, number> = new Map<string, number>();  build(source: string): void {  for (const char of source) {  this.frequencies.set(char, (this.frequencies.get(char) || 0) + 1);  }  this.nodes = Array.from(this.frequencies.entries()).map(([symbol, frequency]) => new TNode(symbol, frequency));  while (this.nodes.length > 1) {  this.nodes.sort((a, b) => a.frequency - b.frequency);  const left = this.nodes.shift() as TNode;  const right = this.nodes.shift() as TNode;  const parent = new TNode('\*', left.frequency + right.frequency);  parent.left = left;  parent.right = right;  this.nodes.push(parent);  this.root = parent;  }  }  encode(source: string): number[] {  const encodedSource: boolean[] = [];  for (const char of source) {  const encodedSymbol = this.root?.traverse(char, []);  if (encodedSymbol) {  encodedSource.push(...encodedSymbol);  }  }  return encodedSource.map(bit => bit ? 1 : 0);  }  decode(bits: number[]): string {  let current = this.root;  let decoded = "";  for (const bit of bits) {  current = bit ? current?.right ?? null : current?.left ?? null;  if (current && this.isLeaf(current)) {  decoded += current.symbol;  current = this.root;  }  }  return decoded;  }  isLeaf(node: TNode): boolean {  return node.left === null && node.right === null;  }  } |

Листинг 1.2 – Класс для построения дерева

|  |
| --- |
| function calculateFrequencyAndProbabilityForEachLetter(message: string) {  const frequencyMap = new Map<string, { frequency: number; probability: number }>();  for (const letter of message) {  const entry = frequencyMap.get(letter) || { frequency: 0, probability: 0 };  entry.frequency++;  entry.probability = entry.frequency / message.length;  frequencyMap.set(letter, entry);  }  const letters = Array.from(frequencyMap.keys());  const frequencies = letters.map(letter => frequencyMap.get(letter)!.frequency);  const probabilities = letters.map(letter => frequencyMap.get(letter)!.probability);  return { letters, frequencies, probabilities };  }  function fillArrsFromLists(letters: string[], frequencies: number[], probabilities: number[]) {  return { sortedLetters: letters, sortedProbabilities: probabilities };  }  function sortArrOfLettersAndProbabilities(sortedLetters: string[], sortedProbabilities: number[]) {  const combined = sortedLetters.map((letter, index) => ({ letter, probability: sortedProbabilities[index] }));  combined.sort((a, b) => b.probability – a.probability);  return { sortedLetters: combined.map(item => item.letter), sortedProbabilities: combined.map(item => item.probability) };  }  function shannonFano(sortedLetters: string[], sortedProbabilities: number[]): string[] {  const letterBits: string[] = Array(sortedLetters.length).fill(‘’);  shannonFanoRecursive(0, sortedLetters.length – 1, sortedProbabilities, letterBits, ‘’);  return letterBits;  }  function shannonFanoRecursive(left: number, right: number, sortedProbabilities: number[], letterBits: string[], prefix: string) {  if (left >= right) {  return;  }  const totalProbability = sortedProbabilities.slice(left, right + 1).reduce((a, b) => a + b, 0);  let sum = 0;  let m = left;  while (sum < totalProbability / 2) {  sum += sortedProbabilities[m];  m++;  }  for (let I = left; I <= right; i++) {  letterBits[i] = I < m ? prefix + ‘0’ : prefix + ‘1’;  }  shannonFanoRecursive(left, m – 1, sortedProbabilities, letterBits, prefix + ‘0’);  shannonFanoRecursive(m, right, sortedProbabilities, letterBits, prefix + ‘1’);  }  function encodeMessage(message: string, sortedLetters: string[], letterBits: string[]): string {  const encodedMessage: string[] = [];  for (const letter of message) {  const letterIndex = sortedLetters.indexOf(letter);  if (letterIndex !== -1) {  encodedMessage.push(letterBits[letterIndex]);  }  }  return encodedMessage.join(‘’);  }  function decodeMessage(encodedMessage: string, sortedLetters: string[], letterBits: string[]): string {  let decodedMessage = ‘’;  let temp = ‘’;  for (const bit of encodedMessage) {  temp += bit;  const bitIndex = letterBits.indexOf(temp);  if (bitIndex !== -1) {  decodedMessage += sortedLetters[bitIndex];  temp = ‘’;  }  }  return decodedMessage;  }  function encodingToBytes(message: string): string {  let bin = ‘’;  for (const char of message) {  const charCode = char.charCodeAt(0);  const binary = charCode.toString(2).padStart(8, ‘0’);  bin += binary;  }  return bin;  } |

Листинг 1.3 – Вспомогательные функции

Код выполнения самих заданий лабораторной работы, листинг 1.4.

|  |
| --- |
| const message = "Timoshenko Dmitry".toLowerCase();  const { letters, frequencies, probabilities } = calculateFrequencyAndProbabilityForEachLetter(message);  const { sortedLetters, sortedProbabilities } = fillArrsFromLists(letters, frequencies, probabilities);  sortArrOfLettersAndProbabilities(sortedLetters, sortedProbabilities);  console.log("Letter Probabilities:");  for (let i = 0; i < sortedLetters.length; i++) {  console.log(` ${sortedLetters[i]}: ${sortedProbabilities[i]}`);  }  console.log("\nShannon-Fano method");  const letterBits = shannonFano(sortedLetters, sortedProbabilities);  console.log("Letter Bits");  for (let i = 0; i < letterBits.length; i++) {  console.log(` ${sortedLetters[i]} ${letterBits[i]}`);  }  const encodedMessage = encodeMessage(message, sortedLetters, letterBits);  console.log(`\nEncoded message: ${encodedMessage}\n`);  const decodedMessage = decodeMessage(encodedMessage, sortedLetters, letterBits);  console.log(`Decoded message: ${decodedMessage}`);  console.log("\n\nHuffman method");  const huffmanTree = new HuffmanTree();  huffmanTree.build(message);  const encoded = huffmanTree.encode(message);  console.log("Encoded message:", encoded.join(''));  const decoded = huffmanTree.decode(encoded);  console.log("\nDecoded message:", decoded);  console.log("\n\nASCII");  const asciiEncoded = encodingToBytes(message);  console.log(`ASCII encoding: ${asciiEncoded}`);  console.log("\n\nRESULT");  console.log(`Shannon-Fano length: ${encodedMessage.length}`);  console.log(`Huffman length: ${encoded.length}`);  console.log(`ASCII length: ${asciiEncoded.length}`); |

Листинг 1.4 – выполнение задания лабораторной работы

Вывод представленного кода, рисунок 1.1.

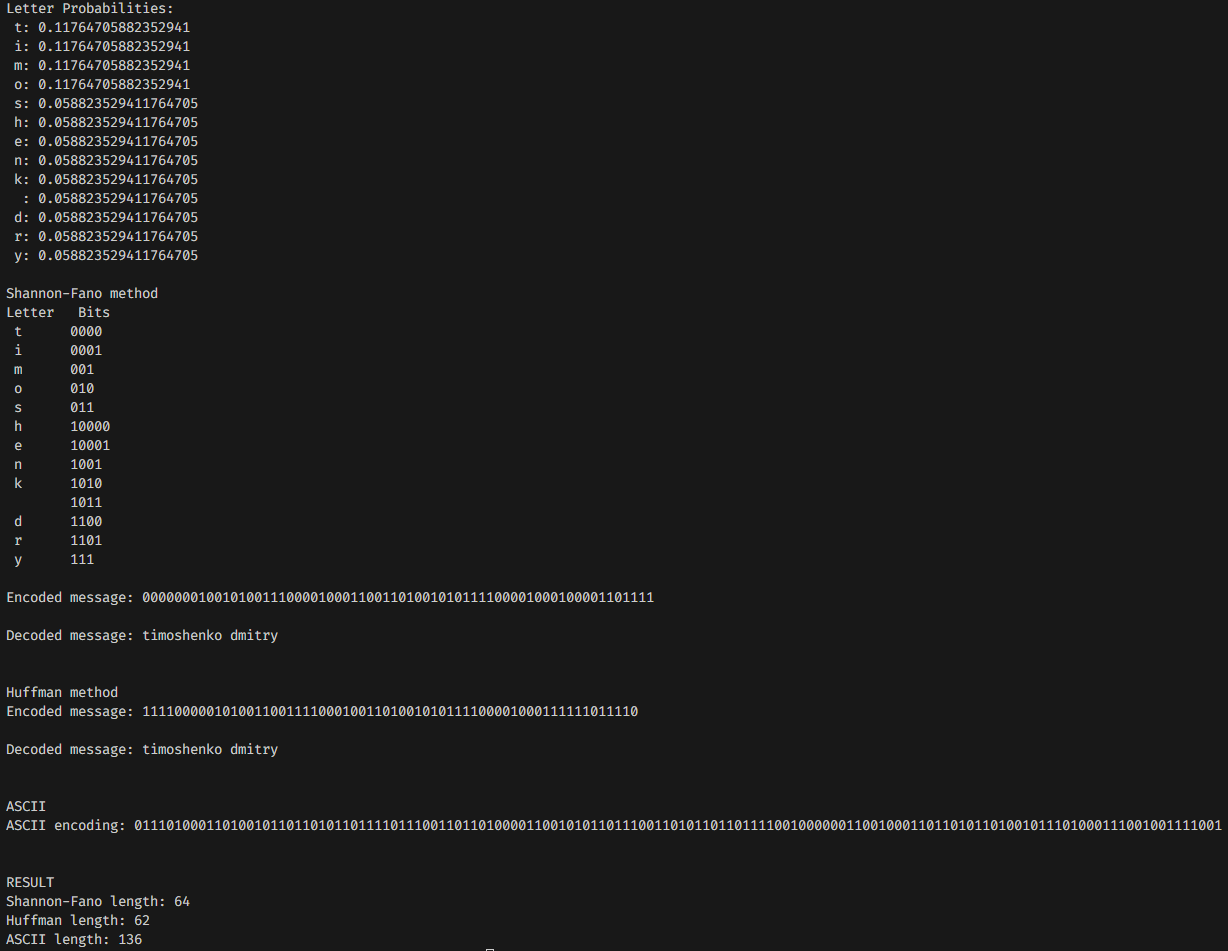


Рисунок 1.1

**Вывод:** в ходе выполнения лабораторной работы были изучены алгоритмы сжатия на основе статических метолов. Были реализованы алгоритмы сжатия/распаковки данных: «Шеннона-Фано» и «Хаффмана». Эффективность в сравнении с кодами ASCII высокая(уменьшение количества бит в 2 раза).