Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра инфокоммуникаций

**ОТЧЕТ**

**ПО ПРАКТИЧЕСКОЙ РАБОТЕ №7**

**дисциплины «Алгоритмизация»**

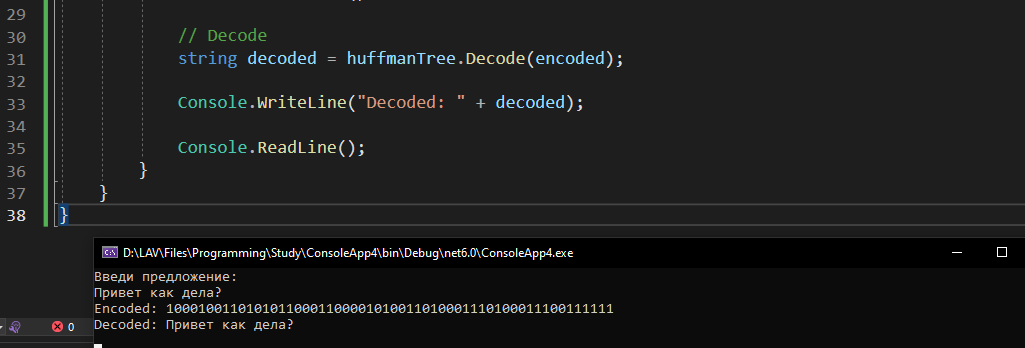
|  |  |  |  |
| --- | --- | --- | --- |
|  | | Выполнил:  Лейс Алексей Вячеславович  2 курс, группа ИВТ-б-о-22-1,  09.03.01 «Информатика и вычислительная техника», направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  | | Руководитель практики кандидат тех. наук доцент кафедры инфокоммуникаций: Воронкин Р.А  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  | |  | |

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_ Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ставрополь, 2023 г.

**Порядок выполнения работы:**

Реализация алгоритма Хаффмана:



Запуск программы и главный алгоритм

Предложение которое ввводит пользователь кодируется по спецальному алгоритму Хаффмана выдавая выходное значение 01.. строку в которой закодированы все буквы.

Код:

*using System;*

*using System.Collections.Generic;*

*using System.Linq;*

*using System.Text;*

*using System.Collections;*

*using System.Xml.Linq;*

*namespace HuffmanTest*

*{*

*public class HuffmanTree*

*{*

*private List<Node> nodes = new List<Node>();*

*public Node Root { get; set; }*

*public Dictionary<char, int> Frequencies = new Dictionary<char, int>();*

*public void Build(string source)*

*{*

*for (int i = 0; i < source.Length; i++)*

*{*

*if (!Frequencies.ContainsKey(source[i]))*

*{*

*Frequencies.Add(source[i], 0);*

*}*

*Frequencies[source[i]]++;*

*}*

*foreach (KeyValuePair<char, int> symbol in Frequencies)*

*{*

*nodes.Add(new Node() { Symbol = symbol.Key, Frequency = symbol.Value });*

*}*

*while (nodes.Count > 1)*

*{*

*List<Node> orderedNodes = nodes.OrderBy(node => node.Frequency).ToList<Node>();*

*if (orderedNodes.Count >= 2)*

*{*

*// Take first two items*

*List<Node> taken = orderedNodes.Take(2).ToList<Node>();*

*// Create a parent node by combining the frequencies*

*Node parent = new Node()*

*{*

*Symbol = '\*',*

*Frequency = taken[0].Frequency + taken[1].Frequency,*

*Left = taken[0],*

*Right = taken[1]*

*};*

*nodes.Remove(taken[0]);*

*nodes.Remove(taken[1]);*

*nodes.Add(parent);*

*}*

*this.Root = nodes.FirstOrDefault();*

*}*

*}*

*public BitArray Encode(string source)*

*{*

*List<bool> encodedSource = new List<bool>();*

*for (int i = 0; i < source.Length; i++)*

*{*

*List<bool> encodedSymbol = this.Root.Traverse(source[i], new List<bool>());*

*encodedSource.AddRange(encodedSymbol);*

*}*

*BitArray bits = new BitArray(encodedSource.ToArray());*

*return bits;*

*}*

*public string Decode(BitArray bits)*

*{*

*Node current = this.Root;*

*string decoded = "";*

*foreach (bool bit in bits)*

*{*

*if (bit)*

*{*

*if (current.Right != null)*

*{*

*current = current.Right;*

*}*

*}*

*else*

*{*

*if (current.Left != null)*

*{*

*current = current.Left;*

*}*

*}*

*if (IsLeaf(current))*

*{*

*decoded += current.Symbol;*

*current = this.Root;*

*}*

*}*

*return decoded;*

*}*

*public bool IsLeaf(Node node)*

*{*

*return (node.Left == null && node.Right == null);*