Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Лабораторная работа № 10

«**ПОСТРОЕНИЕ БИНАРНОГО ДЕРЕВА ПОИСКА.**

**ОБХОДЫ ДЕРЕВА  И  РЕАЛИЗАЦИЯ НА НЕМ ОСНОВНЫХ ОПЕРАЦИЙ С ДАННЫМИ**»

Вариант № 14

Выполнил: студент гр. 981064 Шичко П.В.

Минск 2020

Вариант № 14

Задание № 1.

Ввести произвольные 5-10 символов и задать вероятность появления каждого символа в сообщении.

1. С помощью кодов Хаффмана сжать исходное сообщение. Для этого построить дерево с кодами Хаффмана, затем вывести соответствующие ему двоичные коды каждого символа и определить среднюю длину кода сообщения.

2. Решить задачу декодирования  символов, т.е. показать, как по коду определяется соответствующий ему символ исходного сообщения.

Код программы:

static void Main(string[] args)  
 {  
 Console.WriteLine("Введите исходную строку:");  
 var input = Console.ReadLine();  
 var huffmanTree = new HuffmanTree();  
  
 *// Build the Huffman tree* huffmanTree.Build(input);  
  
 *// Encode* var encoded = huffmanTree.Encode(input);  
  
 Console.Write("Кодирование: ");  
 foreach (bool bit in encoded)  
 {  
 Console.Write((bit ? 1 : 0) + "");  
   
 }  
 Console.WriteLine();  
  
 *// Decode* var decoded = huffmanTree.Decode(encoded);  
  
 Console.WriteLine("Декодирование: " + decoded);  
  
 Console.ReadLine();  
 }  
 }  
 public char Symbol { get; set; }  
 public int Frequency { get; set; }  
 public Node Right { get; set; }  
 public Node Left { get; set; }  
  
 public List<bool> Traverse(char symbol, List<bool> data)  
 {  
 *// Leaf* if (Right == null && Left == null)  
 {  
 return symbol.Equals(this.Symbol) ? data : null;  
 }  
  
 List<bool> left = null;  
 List<bool> right = null;  
  
 if (Left != null)  
 {  
 var leftPath = new List<bool>();  
 leftPath.AddRange(data);  
 leftPath.Add(false);  
  
 left = Left.Traverse(symbol, leftPath);  
 }  
  
 if (Right != null)  
 {  
 var rightPath = new List<bool>();  
 rightPath.AddRange(data);  
 rightPath.Add(true);  
 right = Right.Traverse(symbol, rightPath);  
 }  
  
 return left ?? right;  
 }  
}  
public class HuffmanTree  
{  
 private readonly List<Node> \_nodes = new List<Node>();  
 private Node Root { get; set; }  
 private readonly Dictionary<char, int> \_frequencies = new Dictionary<char, int>();  
  
 public void Build(string source)  
 {  
 foreach (var t in source)  
 {  
 if (!\_frequencies.ContainsKey(t))  
 {  
 \_frequencies.Add(t, 0);  
 }  
  
 \_frequencies[t]++;  
 }  
  
 foreach (var (key, value) in \_frequencies)  
 {  
 \_nodes.Add(new Node() { Symbol = key, Frequency = value });  
 }  
  
 while (\_nodes.Count > 1)  
 {  
 var orderedNodes = \_nodes.OrderBy(node => node.Frequency).ToList<Node>();  
  
 if (orderedNodes.Count >= 2)  
 {  
 *// Take first two items* var taken = orderedNodes.Take(2).ToList<Node>();  
  
 *// Create a parent node by combining the frequencies* var 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);  
 }  
  
 Root = \_nodes.FirstOrDefault();  
 }  
 }  
  
 public BitArray Encode(string source)  
 {  
 var encodedSource = new List<bool>();  
  
 foreach (var t in source)  
 {  
 var encodedSymbol = this.Root.Traverse(t, new List<bool>());  
 encodedSource.AddRange(encodedSymbol);  
 Console.WriteLine($"Символ - {t}, код - {Output(encodedSymbol)}");  
 }  
  
 var bits = new BitArray(encodedSource.ToArray());  
  
 return bits;  
 }  
  
 private static string Output(IEnumerable<bool> encodedSymbol)  
 {  
 var str="";  
 foreach (var t in encodedSymbol)  
 {  
 if (t)  
 {  
 str += '1';  
 }  
 else str += '0';  
 }  
 return str;  
 }  
  
 public string Decode(BitArray bits)  
 {  
 var current = this.Root;  
 var 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;  
 }  
  
 private static bool IsLeaf(Node node)  
 {  
 return (node.Left == null && node.Right == null);  
 }

