|  |  |  |  |
| --- | --- | --- | --- |
| МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  **Кафедра програмних систем і технологій**  Дисципліна  **«Структури даних, аналіз і алгоритми комп'ютерної обробки інформації»**  **«Код Хаффмана»** | | | |
| **Виконав**: | Самойленко Олександр Васильович | **Перевірив**: | Бичков Олексій Сергійович |
| **Група** | ІПЗ-21 | **Дата перевірки** |  |
| **Форма навчання** | денна | **Оцінка** |  |
| **Спеціальність** | 121 |
| 2022 | | | |

# Завдання:

1. Написати програму реалізації алгоритму Хаффмана

# Аналіз алгоритмів:

Для побудови коду Хаффмана спочатку занайдемо частоти всіх символів вхідної стрічки. Зберігатимемо їх у словнику. Після цього створимо ноди бінарного дерева коду, міститимуть в собі символ, його частоту та посилання на нащадків. Далі «пов’язуватимемо» між собою 2 елементи з найменшою частотою шляхом створення нового, батьківського для кожної з них, вузла. Новий вузол міститиме null як дані та суму частот нащадків як частоту. Після цього видалятимемо 2 «оброблені» ноди з загального списку та додаватимемо новостворений вузол. Повторюватимемо доти, доки у списку не залишиться 1 елемент – корінь дерева.

Кодування символа визначатимемо за класичною схемою коду Хаффмана: праворуч-1, ліворуч-0.

# Код програми:

public class HuffmanNode  
{  
 public char? Data { get; set; }  
 public int Frequency { get; set; }  
 public HuffmanNode? Left { get; set; }  
 public HuffmanNode? Right { get; set; }  
  
 public string? GetCode(char symbol, string binaryCode = "")  
 {  
 if (Data == symbol)  
 {  
 return binaryCode;  
 }  
 if (Left is not null)  
 {  
 var path = Left.GetCode(symbol, binaryCode + "0");  
 if (path is not null)  
 {  
 return path;  
 }  
 }  
 if (Right is not null)  
 {  
 var path = Right.GetCode(symbol, binaryCode + "1");  
 if (path is not null)  
 {  
 return path;  
 }  
 }  
 return null;  
 }  
}

public class HuffmanCode  
{  
 public string? OriginalString { get; private set; }  
 public string EncodedString => GetEncodedString(OriginalString, Codes);  
 public string DecodedString => DecodeString(EncodedString, \_parentNode);  
 private readonly HuffmanNode \_parentNode;  
 public IReadOnlyDictionary<char, string?> Codes { get; private set; }  
 public HuffmanCode(string? inputString)  
 {  
 OriginalString = inputString;  
 var symbolsFrequencies = GetSymbolsFrequencies(inputString);  
 var nodeList = ConstructNodes(symbolsFrequencies);  
 \_parentNode = BuildTree(nodeList);  
 Codes = GetCodes(symbolsFrequencies, \_parentNode);  
 }  
 private static List<HuffmanNode> ConstructNodes(Dictionary<char, int> symbols)  
 {  
 var nodes = symbols.Select(symbol => new HuffmanNode()  
 {  
 Data = symbol.Key,   
 Frequency = symbol.Value  
 })  
 .ToList();  
 return nodes;  
 }  
  
 private static Dictionary<char, int> GetSymbolsFrequencies(string? inputLine)  
 {  
 if (inputLine is null)  
 return new Dictionary<char, int>();  
 var symbolFrequencies = inputLine  
 .GroupBy(c => c)  
 .Select(g => new  
 {  
 Key = g.Key,  
 Freq = g.Count()  
 })  
 .ToDictionary(  
 keySelector: keyFreqPair => keyFreqPair.Key,   
 elementSelector: keyFreqPair => keyFreqPair.Freq);  
   
 return symbolFrequencies;  
 }  
  
 private static HuffmanNode BuildTree(List<HuffmanNode> treeNodes)  
 {  
 while (treeNodes.Count > 1)  
 {  
 treeNodes = treeNodes  
 .OrderByDescending(node => node.Frequency)  
 .ToList();  
 var parent = new HuffmanNode()  
 {  
 Data = null,   
 Frequency = treeNodes[^2].Frequency + treeNodes[^1].Frequency,   
 Left = treeNodes[^2],   
 Right = treeNodes[^1]  
 };  
 treeNodes = treeNodes   
 .Take(treeNodes.Count - 2) *//All except two last* .ToList();  
 treeNodes.Add(parent);  
  
 }  
  
 return treeNodes.First();  
 }  
  
 private static Dictionary<char, string?> GetCodes (IReadOnlyDictionary<char, int> symbols, HuffmanNode tree)  
 {  
 var codes = symbols  
 .ToDictionary(  
 keySelector: symbol => symbol.Key,   
 elementSelector: symbol => tree.GetCode(symbol.Key));  
 return codes;  
 }  
  
 private static string GetEncodedString(string s, IReadOnlyDictionary<char, string?> codes)  
 {  
 var encoded = string.Empty;  
 foreach (var c in s)  
 {  
 encoded += codes[c];  
 }  
  
 return encoded;  
 }  
  
 private static string DecodeString(string encoded, HuffmanNode root)  
 {  
 var decoded = string.Empty;  
  
 var current = root;  
 foreach (var c in encoded)  
 {  
 if (c == '0')  
 current = current?.Left;  
 else  
 current = current?.Right;  
  
 if (current?.Data is null)  
 {  
 continue;  
 }  
 decoded += current.Data;  
 current = root;  
 }  
  
 return decoded;  
 }  
}

# Результат виконання:



# Висновки:

Було розроблено та протестовано алгоритм кодування Хаффмана