|  |  |
| --- | --- |
| Министерство образования Республики Беларусь  Учреждение образования «Полоцкий государственный университет» | |
|  | Факультет информационных технологий  Кафедра технологий программирования |
| Лабораторная работа №1 по курсу «Теория информации»  «Сжатие без потерь» | |
| Выполнил | Студент гр. 21-ИТ-1  Макеёнок Д.И. |
| Проверила | Васильева Д.М. |
| Полоцк, 2023г. | |

**Ход работы**

Метод оптимального побуквенного кодирования был разработан в 1952 г. Д. Хаффманом. Оптимальный код Хаффмана обладает минимальной средней длиной кодового слова среди всех побуквенных кодов для данного источника с алфавитом *А=*{*a*1*,…,an*} и вероятностями *pi =P*(*ai*).

**Задание:**

1. Изучить теоретический материал
2. Реализовать процедуру построения оптимального кода Хаффмана.
3. Построить код Хаффмана для текста на языке, обозначенном преподавателем, использовать файл не менее 1 Кб. Распечатать полученную кодовую таблицу в виде:

|  |  |  |  |
| --- | --- | --- | --- |
| Символ | Вероятность | Кодовое слово | Длина кодового слова |
|  |  |  |  |

1. Проверить выполнение неравенства Крафта-МакМиллана для полученного кода
2. Вычислить энтропию исходного файла и сравнить со средней длиной кодового слова.
3. Восстановить исходный текст

Листинг 1 - реализация программы:

static class Program

{

static string Result = string.Empty;

private static void Main()

{

string n = File.ReadAllText("../../../save.txt");

Console.WriteLine($"Entropy: {Entropy(n)}");

Dictionary<char, int> letters = new();

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

{

if (!letters.ContainsKey(n[i]))

letters.Add(n[i], n.Count(x => x == n[i]));

}

Dictionary<Node, int> sorted = new();

foreach (var c in letters.OrderBy(k => k.Value))

{

sorted.Add(new Node(c.Key), c.Value);

}

Dictionary<Node, int> saveNodes = new();

if (sorted.Count == 1)

{

var s = sorted.First();

s.Key.Code = "0";

saveNodes.Add(s.Key, s.Value);

}

while (sorted.Count != 1)

{

var left = MinBy(sorted);

sorted.Remove(left.Key);

var right = MinBy(sorted);

sorted.Remove(right.Key);

var temp = new Node('t');

temp.AddNeighbour(left.Key, right.Key);

sorted.Add(temp, left.Value + right.Value);

left.Key.AddToCode("1");

right.Key.AddToCode("0");

if (left.Key.Value is not null)

{

saveNodes.Add(left.Key, left.Value);

}

if (right.Key.Value is not null)

{

saveNodes.Add(right.Key, right.Value);

}

sorted = SortByValue(sorted);

}

List<double> asd = new List<double>();

int dd = n.Length;

foreach (var c in letters)

{

int let\_counter = 0;

foreach(char ch in n)

{

if (ch == c.Key) let\_counter++;

}

asd.Add(let\_counter);

}

Dictionary<char, string> map = new();

foreach (var sn in saveNodes.OrderByDescending(x => x.Value))

{

map.Add(sn.Key.VAL, sn.Key.Code);

}

string res = String.Empty;

foreach (var v in n)

{

res += map[v];

}

Console.WriteLine($"Symbol\tCode\tLength\tProbability");

int j = 0;

foreach (var sn in saveNodes.OrderByDescending(x => x.Value))

{

Console.WriteLine($"{sn.Key.Value}\t{sn.Key.Code}\t{sn.Key.Code.Length}\t{Math.Round(asd[j]/dd, 4)}");

j++;

}

Console.WriteLine(res);

string decoded = String.Empty;

string token = string.Empty;

foreach (var ch in res)

{

token += ch;

if (map.ContainsValue(token))

{

decoded += map.FirstOrDefault(x => x.Value == token).Key;

token = String.Empty;

}

}

Console.WriteLine(decoded);

}

private class Node

{

public char? Value;

public Node? Left, Right;

public char VAL;

public string Code = string.Empty;

public Node(char value)

{

Value = value;

VAL = value;

}

public void AddNeighbour(Node left, Node right)

{

Value = null;

Left = left;

Right = right;

}

public void AddToCode(string dir)

{

Code = dir + Code;

Left?.AddToCode(dir);

Right?.AddToCode(dir);

}

}

private static Dictionary<Node, int> SortByValue(Dictionary<Node, int> letters)

{

Dictionary<Node, int> sorted = new();

foreach (var c in letters.OrderBy(k => k.Value))

{

sorted.Add(c.Key, c.Value);

}

return sorted;

}

private static KeyValuePair<Node, int> MinBy(Dictionary<Node, int> letters)

{

Dictionary<Node, int> sorted = new();

foreach (var c in letters.OrderBy(k => k.Value))

{

return new KeyValuePair<Node, int>(c.Key, c.Value);

}

return new KeyValuePair<Node, int>();

}

public static int Entropy(this string s)

{

HashSet<char> chars = new HashSet<char>(s);

return chars.Count;

}

}

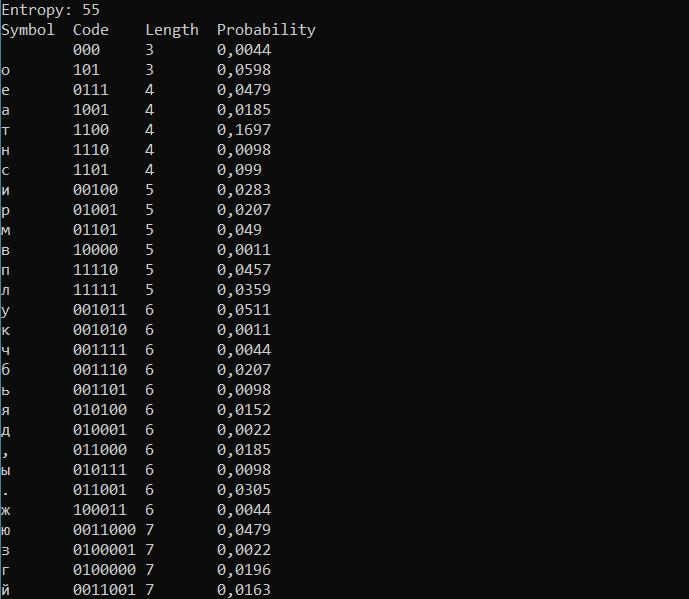


Рисунок 1. Кодовая таблица и энтропия

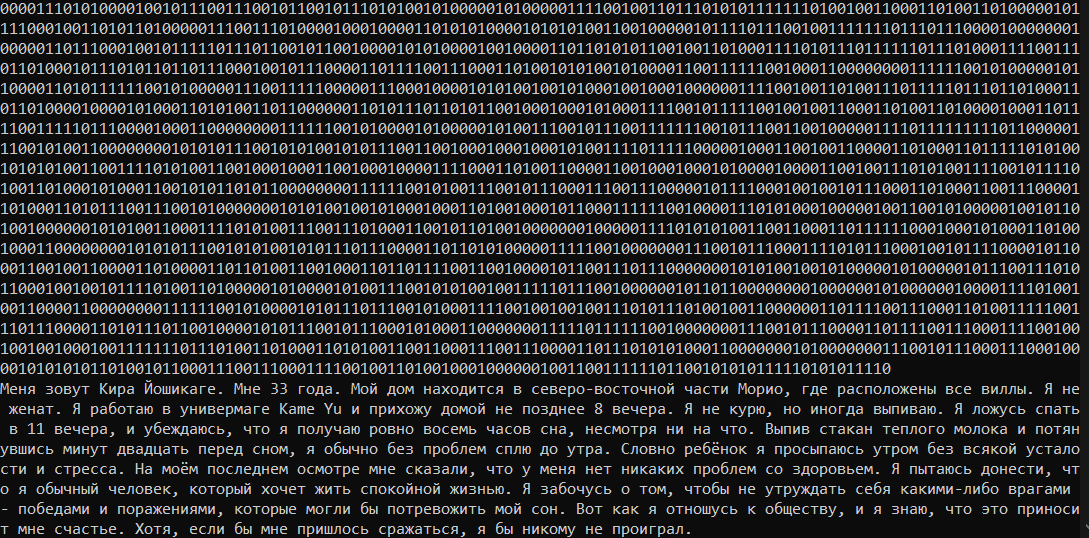


Рисунок 2. Код и расшифрованный текст