МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа № 1**

по дисциплине: Теория информации

тема: «Исследование кодирования по методу Хаффмана. Оценка эффективности кода»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

Твердохлеб Виталий Викторович

Белгород 2024г.

**Лабораторная работа №1**

**«Исследование кодирования по методу Хаффмана. Оценка эффективности кода»**

**Цель работы:** Исследование кодирования по методу Хаффмана. Оценка эффективности кода.

**Решение задач:**

**Вариант 3**

1. Построить кодовое представление сообщения, вероятности появления символов в пределах алфавита которого приведены в табл.1.

Изображение выглядит как диаграмма, линия

Автоматически созданное описание

S8 – 00000

S7 – 10000

S6 – 1000

S5 – 100

S4 – 001

S3 – 101

S2 – 10

S1 – 11

2. Построить кодовое представление сообщения, вероятности появления символов в пределах алфавита которого приведены в табл.2.

Изображение выглядит как диаграмма, линия

Автоматически созданное описание

S8 – 00000

S7 – 10000

S6 – 1000

S5 – 001

S4 – 101

S3 – 100

S2 – 11

S1 – 10

3. Построить кодовое представление сообщения:

оитомии о ими оооитми о о о ооиимтомиимотоим оои тоо и и м оио иомтоо тоимо т и

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Символ | 0 | и | пробел | м | Т |
| Вероятность | 0.31 | 0.24 | 0.22 | 0.13 | 0.10 |

Изображение выглядит как диаграмма, линия

Автоматически созданное описание

Т – 000

М – 100

Пробел – 10

И – 01

О – 11

Кодовое представление сообщения:

1110010110111010001100100111000111111100100111000110011001100111110100110101101110100111101011100110011111000010111100100010000110011101100100011011010111100010111001111000100010

4. Для условий, приведенных в заданиях 1 и 2 и 3, выявить возможность построения альтернативных кодовых моделей сообщения. В случае обнаружения таковых, выявить наиболее эффективные из них по критериям.

Вычисление коэффициента сжатия и дисперсии

**Задание 3:**

Т – 000, М – 100, пробел – 10, И – 01, О – 11.

n=79; η=3;

B=79\*3=237;

=176;

Коэффициент сжатия :

Средняя длинна кода L.

Значение дисперсии:

5. Алгоритм Хаффмана на языке Python

import heapq  
from collections import defaultdict  
  
def buildHuffmanTree(text):  
 # Подсчет частоты символов текста в словаре  
 symbols\_freq = defaultdict(int)  
 for symbol in text:  
 symbols\_freq[symbol] += 1  
  
 # Строим список [<"вес" символа>, [<символ>, <место для кода>]] из словаря  
 heap = [[weight, [symbol, ""]] for symbol, weight in symbols\_freq.items()]  
 # Преобразуем список в кучу  
 heapq.heapify(heap)  
  
 while len(heap) > 1:  
 # Удаляем два наименьших элемента из кучи  
 left = heapq.heappop(heap)  
 right = heapq.heappop(heap)  
  
 # Формируем коды для двух значений  
 for pair in left[1:]:  
 pair[1] = '0' + pair[1]  
  
 for pair in right[1:]:  
 pair[1] = '1' + pair[1]  
  
 # Добавляем результирующий "вес" в кучу  
 heapq.heappush(heap, [left[0] + right[0]] + left[1:] + right[1:])  
  
 return heap[0]  
  
# Перевод получившегося результата из кучи в список  
def buildHuffmanCodes(tree):  
 huff\_codes = {}  
  
 for pair in tree[1:]:  
 symbol, code = pair  
 huff\_codes[symbol] = code  
  
 return huff\_codes

Работа на примере задачи 3:

text = "оитомии о ими оооитми о о о ооиимтомиимотоим оои тоо и и м оио иомтоо тоимо т и"  
  
# Построение дерева Хаффмана и генерация кодов  
huffmanTree = buildHuffmanTree(text)  
huffmanCodes = buildHuffmanCodes(huffmanTree)  
  
print(f"{text}")  
print(f"{huffmanCodes}")

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Функция нахождения коэффициента сжатия и избыточности кода

def printCompressionRatioAndRedundacy(text):  
 different\_symbols\_num = symbolsNum(text)  
 all\_symbols\_num = sum([value for value in different\_symbols\_num.values()])  
  
 eta = 1  
 while eta \* eta < len(different\_symbols\_num):  
 eta += 1  
  
 b = eta \* all\_symbols\_num  
  
 huffmanTree = buildHuffmanTree(text)  
 huffmanCodes = buildHuffmanCodes(huffmanTree)  
  
 bst = 0  
 for key in different\_symbols\_num:  
 if key in huffmanCodes:  
 bst += different\_symbols\_num[key] \* len(huffmanCodes[key])  
  
 k = b / bst  
  
 for num in different\_symbols\_num:  
 different\_symbols\_num[num] = different\_symbols\_num[num] / all\_symbols\_num  
  
 l = 0  
 for key in different\_symbols\_num:  
 l += different\_symbols\_num[key] \* len(huffmanCodes[key])  
  
 delta = 0  
 for key in different\_symbols\_num:  
 delta += different\_symbols\_num[key] \* math.pow(len(huffmanCodes[key]) - l, 2)  
  
 print(f"Compression: {k}")  
 print(f"Redundancy: {delta}")

Работа на примере задачи 3:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

**Вывод:** в ходе выполнения лабораторной работы исследовано кодирование по методу Хаффмана. Оценена эффективность кода.