Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Сибирский государственный университет телекоммуникаций и информатики»  
(СибГУТИ)

Институт информатики и вычислительной техники  
Кафедра прикладной математики и кибернетики

**Практическая работа №3  
 по дисциплине «Теория информации»  
на тему «Блочное кодирование»**

Выполнили:  
студенты гр.ИП-014

Обухов А.И.

Проверила:  
Старший преподаватель каф. ПМиК  
Дементьева Кристина Игоревна

Новосибирск 2024 г.

**Цель работы:** Экспериментальное изучение свойств блочного кодирования.

**Язык программирования:** С, С++, С#, Python

**Результат:** программа, тестовые примеры, отчет.

**Задание:**

1. Для выполнения работы необходим сгенерированный файл с неравномерным распределением из практической работы 1.

При блочном кодировании входная последовательность разбивается на блоки равной длины, которые кодируются целиком. Поскольку вероятностное распределение символов в файле известно, то и вероятности блоков могут быть вычислены и использованы для построения кода.

2. Закодировать файл блочным методом кодирования (можно использовать любой метод кодирования), размер блока . Вычислить избыточность кодирования на символ входной последовательности для каждого размера блока.

3. После тестирования программы необходимо заполнить таблицу и проанализировать полученные результаты, сравнить с теоретическими оценками.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Длина блока n=1 | Длина блока n=2 | Длина блока n=3 | Длина блока n=4 |
| Оценка избыточности кодирования на один символ входной последовательности | 0.05323 | 0.03636 | 0.03239 | 0.03009 |

**Скриншоты работы программы:**  
****

**Анализ результатов работы программы**

В ходе работы были получены избыточности при кодировании блоков разной длины:

1. При размере блока n=1 избыточность составляет 0.05323
2. При размере блока n=2 избыточность составляет 0.03636
3. При размере блока n=3 избыточность составляет 0.03239
4. При размере блока n=4 избыточность составляет 0.03009

Таким образом, увеличение размера блоков в кодировании приводит к снижению избыточности на один символ входной последовательности. Это происходит из-за увеличения количества символов, на которые распространяется кодирование. В результате, требуется меньшее количество бит для кодирования символов, что снижает избыточность.

Теоретические оценки подтверждают, что с ростом размера блоков избыточность снижается, как и показали наши эксперименты.

Таким образом, использование больших размеров блоков в кодировании может быть более эффективным, особенно при работе с большими объемами данных.

**Листинг программы**

import heapq

from collections import Counter

import math

from lab1 import \*

from lab2 import Node

from typing import Final

FILE\_LENGTH: Final[int] = 50\_000

PROBABILITIES: Final[dict[str, float]] = {'a': 0.3, 'b': 0.2, 'c': 0.1, 'd':0.4}

BLOCK\_SIZES: Final[list[int]] = list(range(1, 5))

def generate\_block\_partitions(sequence, block\_sizes: list[int]) -> list[list[str]]:

partitions = []

for size in block\_sizes:

partitions.append([sequence[i:i+size] for i in range(0, len(sequence), size)])

return partitions

def calculate\_entropy\_by\_block\_size(text: str, block\_size: int) -> float:

blocks = [text[i:i+block\_size] for i in range(0, len(text), block\_size)]

freqs = Counter(blocks)

probs = {k: v / len(blocks) for k, v in freqs.items()}

entropy = -sum(p \* math.log2(p) for p in probs.values() if p != 0)

return entropy

def huffman\_encode(line: str, block\_size: int) -> None:

def build\_huffman\_tree(text):

char\_freq = Counter(text)

heap = [Node(char, freq) for char, freq in char\_freq.items()]

heapq.heapify(heap)

while len(heap) > 1:

left = heapq.heappop(heap)

right = heapq.heappop(heap)

merged = Node(None, left.freq + right.freq)

merged.left = left # type:ignore

merged.right = right # type:ignore

heapq.heappush(heap, merged)

return heap[0]

def build\_huffman\_codes(node, prefix="", codes={}):

if node:

if node.char is not None:

codes[node.char] = prefix

build\_huffman\_codes(node.left, prefix + '0', codes)

build\_huffman\_codes(node.right, prefix + '1', codes)

split\_line = [line[i: i + block\_size] for i in range(0, len(line), block\_size)]

probabilities = {k: v / len(split\_line) for k, v in Counter(split\_line).items()}

probabilities = dict(sorted(probabilities.items(), key=lambda item: item[1], reverse=True))

root = build\_huffman\_tree(split\_line)

codes = {}

build\_huffman\_codes(root, "", codes)

for i in probabilities.keys():

print(f"{i}: {probabilities.get(i, .0):.5f} - {codes.get(i, '')}")

l\_average = sum(probabilities[i] \* len(codes.get(i, '')) for i in probabilities.keys())

print(f"Средняя длина кодового слова (L average) = {l\_average:.5f}")

entropy = calculate\_entropy\_by\_block\_size(line, block\_size)

print(f'H({block\_size}) = {entropy:.5f}')

r\_h = l\_average - entropy

print(f"R({block\_size}) = {r\_h:.5f}")

def main() -> int:

generate\_file('./input/diff\_prob.txt', PROBABILITIES, FILE\_LENGTH)

input\_text = preprocess\_file('./input/diff\_prob.txt', 'en')

orig\_entropy = calc\_entropy(input\_text, 1)

print(f"H = {orig\_entropy[0]}\np = {orig\_entropy[1]}")

blocks = generate\_block\_partitions(input\_text, BLOCK\_SIZES)

for size, block\_list in zip(BLOCK\_SIZES, blocks):

print(f"Размер блока {size}: {len(block\_list)}")

huffman\_encode(''.join(block\_list), size)

print()

return 0

if \_\_name\_\_ == "\_\_main\_\_":

exit(main())