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

РГЗ

по дисциплине: Теория информации

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович

Проверил:
Твердохлеб Виталий Викторович

Белгород 2024г.

Постановка задачи

Выбрать сочетание источник-кодер-помехоустойчивость(опционально) - характер помех и построить модель канала.

На приёмной стороне реконструировать данные и отобразить результат, проанализировать.

Ход работы

Источник:

Источником был выбран алгоритм Хартли. Он генерирует строки, в которых каждый символ алфавита имеет одинаковую вероятность появления. Это создаёт равномерное распределение символов, что полезно для тестирования алгоритмов в условиях, где нет очевидных предрасположенностей к определённым символам. Информационная энтропия для источника Хартли определяется формулой:

$$H_0 = \log_2 N, \quad (1)$$

где N – количество различных символов в алфавите.

Кодирование:

Для преобразования сообщений был выбран метод кодирования Хаффмана. Этот метод основывается на принципе, согласно которому часто встречающиеся символы кодируются более короткими битовыми последовательностями, а редко встречающиеся символы — более длинными. Он особенно эффективен при неравномерном распределении символов, поскольку позволяет сократить общую длину сообщения за счёт использования более коротких кодов для символов, встречающихся чаще.

Алгоритм работы кодирования Хаффмана:

1. Построение частотной карты.
2. Создание приоритетной очереди из узлов, отсортированных по частоте.
3. Построение дерева Хаффмана путем объединения узлов с наименьшей частотой.
4. Создание кодировок путем обхода дерева от корня до листьев.

Формула для средней длины кодового слова:

$$L = \sum_{i=1}^{n} p_i \cdot l_i, \quad (2)$$

где l_i – длина кодового слова для символа.

Реализация:

Класс HuffmanNode представляет узел дерева Хаффмана, который используется для кодирования символов. Он хранит символ, частоту символа, а также ссылки на левый и правый потомки. Реализацию можно увидеть на «листинг 1».

Листинг 1 - Реализация класса HuffmanNode

```
import random
import string

# Класс узла дерева Хаффмана
# Этот класс представляет узел дерева, который используется для кодирования символов.
class HuffmanNode:
    def __init__(self, ch, frequency, left, right):
        self.ch = ch  # Символ, который представляет этот узел.
        self.frequency = frequency  # Частота символа в исходной строке.
        self.left = left  # Левый потомок узла.
        self.right = right  # Правый потомок узла.

    def __str__(self):
        return "(" + str(self.ch) + ", " + str(self.frequency) + ")"
```

Класс HuffmanCoding содержит методы для построения дерева Хаффмана, создания кодов, а также для кодирования и декодирования строк. Реализацию можно увидеть на «листинг 2»

Листинг 2 - Реализация класса HuffmanCoding и его функций GetCode, buildFrequencyMap, sortByFrequence, buildTree, createHuffmanCode, createCodeRecurse, encode, decode

```
# Класс для кодирования Хаффмана
# Этот класс содержит методы для построения дерева Хаффмана, создания кодов и кодирования/декодирования строки.
class HuffmanCoding:
    # Метод для получения кода Хаффмана
    # Основной метод, который управляет процессом создания кодировки Хаффмана.
    def getCode(self, inputString):
        freqMap = self.buildFrequencyMap(inputString)  # Построение частотной карты символов.
```

```
        nodeQueue = self.sortByFrequence(freqMap)  # Сортировка узлов по ча-
стоте.
        self.root = self.buildTree(nodeQueue)  # Построение дерева Хаффмана.
        codeMap = self.createHuffmanCode(self.root)  # Создание кодировки
Хаффмана.
        return codeMap

    # Метод для построения частотной карты
    # Создает словарь, где ключами являются символы, а значениями — их ча-
стоты в исходной строке.
    def buildFrequencyMap(self, inputString):
        freqMap = {}
        for c in inputString:
            freqMap[c] = freqMap.get(c, 0) + 1
        return freqMap

    # Метод для сортировки узлов по частоте
    # Преобразует частотную карту в список узлов и сортирует их по частоте.
    def sortByFrequence(self, frequencyMap):
        queue = []
        for k, v in frequencyMap.items():
            queue.append(HuffmanNode(k, v, None, None))
        queue.sort(key=lambda x: x.frequency)
        return queue

    # Метод для построения дерева Хаффмана
    # Объединяет два узла с наименьшей частотой до тех пор, пока не останется
один узел.
    def buildTree(self, nodeQueue):
        while len(nodeQueue) > 1:
            node1 = nodeQueue.pop(0)
            node2 = nodeQueue.pop(0)
            node = HuffmanNode('', node1.frequency + node2.frequency, node1,
node2)
            nodeQueue.append(node)
            nodeQueue.sort(key=lambda x: x.frequency)
        return nodeQueue.pop(0)

    # Метод для создания кодировки Хаффмана
    # Создает словарь, в котором каждому символу соответствует его код Хафф-
мана.
    def createHuffmanCode(self, root):
        codeMap = {}
        self.createCodeRecurse(root, codeMap, "")
        return codeMap

    # Рекурсивный метод для создания кодировок
    # Рекурсивно создает коды для символов, добавляя '0' при переходе влево и
'1' при переходе вправо.
    def createCodeRecurse(self, node, frequencyMap, s):
        if node.left is None and node.right is None:
            frequencyMap[node.ch] = s
            return
        self.createCodeRecurse(node.left, frequencyMap, s + '0')
        self.createCodeRecurse(node.right, frequencyMap, s + '1')

    # Метод для кодирования строки
    # Кодирует входную строку, заменяя каждый символ на соответствующий код
Хаффмана.
    def encode(self, codeMap, inputString):
        s = ""
        for i in range(len(inputString)):
            s += codeMap.get(inputString[i])
        return s
```

```
    # Метод для декодирования строки
    # Декодирует закодированную строку, проходя по дереву Хаффмана на основе
битов закодированной строки.
    def decode(self, coded):
        s = ""
        curr = self.root
        for i in range(len(coded)):
            curr = curr.right if coded[i] == '1' else curr.left
            if curr.left is None and curr.right is None:
                s += curr.ch
                curr = self.root
        return s
```

Функция hartleySource генерирует случайную строку заданной длины из символов латинского алфавита. Реализацию можно увидеть на «листинг 3»

Листинг 3 - Реализация функции hartleySource

```
# Функция для генерации случайной строки на основе алфавита Хартли
# Генерирует случайную строку заданной длины из символов латинского алфавита.
def hartleySource(length):
    return ''.join(random.choice(string.ascii_lowercase) for _ in
range(length))
```

Функция splitStringIntoPairs разбивает строку на пары символов, чтобы усложнить задачу кодирования и продемонстрировать работу алгоритма на более сложных данных. Реализацию можно увидеть на «листинг 4»

Листинг 4 - Реализация функции splitStringIntoPairs

```
# Функция для разделения строки на пары символов
# Разбивает строку на пары символов, чтобы усложнить задачу кодирования и
продемонстрировать работу алгоритма на более сложных данных.
def splitStringIntoPairs(s):
    return [s[i:i + 2] for i in range(0, len(s), 2)]
```

Функция stringToBinary преобразует строку в последовательность битов. Реализацию можно увидеть на «листинг 5»

Листинг 5 - Реализация функции stringToBinary

```
# Функция для перевода строки в двоичный вид
# Преобразует строку в последовательность битов.
def stringToBinary(s):
    return ''.join(format(ord(c), '08b') for c in s)
```

Пример работы программы:

Основная функция main управляет процессом ввода строки, её кодирования и декодирования, а также выводит результаты. Реализацию можно увидеть на «листинг 6», а примеры работы на «рисунок 1» и «рисунок 2».

Листинг 6 - Использование разработанных классов

```
def main():
    choice = input("Введите «1» для случайной генерации или «2» для ручного
ввода: ")

    if choice == '1':
        length = int(input("Введите длину случайной строки: "))
        inputString = hartleySource(length)
    elif choice == '2':
        inputString = input("Введите свою строку: ")
    else:
        print("Неверный выбор!")
        return

    # Переводим исходную строку в двоичный вид и считаем длину
    binary_string = stringToBinary(''.join(inputString))
    original_length_bits = len(binary_string)
    print(f"Длина исходной строки: {original_length_bits}")

    # Разбиваем строку на пары символов
    inputString = splitStringIntoPairs(inputString)  # Это позволяет услож-
нить задачу и продемонстрировать работу алгоритма на более сложных данных.
    print(f"Исходная строка: {inputString}")

    huffman = HuffmanCoding()
    codeMap = huffman.getCode(inputString)  # Получаем кодировку Хаффмана.
    print(f"Код: {str(codeMap)}")

    encoded = huffman.encode(codeMap, inputString)  # Кодируем строку.
    print(f"Закодированное сообщение: {encoded}")
    print(f"Длина закодированного сообщения: {len(encoded)}")

    decoded = huffman.decode(encoded)  # Декодируем сообщение.
    print(f"Декодированная строка: {decoded}")

if __name__ == "__main__":
    main()
```

Пример №1

```
Введите «1» для случайной генерации или «2» для ручного ввода: 1
Введите длину случайной строки: 50
Длина исходной строки: 400
Исходная строка: ['qx', 'xr', 'hq', 'lw', 'tm', 'hw', 'md', 'he', 'rm',
 'wh', 'ta', 'co', 'oo', 'on', 'fj', 'vg', 'lo', 'mn', 'bt', 'bg', 'sm',
  'em', 'rp', 'tk', 'ex']
Код: {'bt': '0000', 'bg': '0001', 'sm': '0010', 'em': '0011', 'rp':
 '0100', 'tk': '0101', 'ex': '0110', 'qx': '01110', 'xr': '01111', 'hq':
  '10000', 'lw': '10001', 'tm': '10010', 'hw': '10011', 'md': '10100',
 'he': '10101', 'rm': '10110', 'wh': '10111', 'ta': '11000', 'co':
 '11001', 'oo': '11010', 'on': '11011', 'fj': '11100', 'vg': '11101',
 'lo': '11110', 'mn': '11111'}
Закодированное сообщение:
 01110011111000010001100101001110100101011011010111110001100111010110111
 11001110111110111110000000100100011010001010110
Длина закодированного сообщения: 118
Декодированная строка: qxxrhqlwtmhwmdhermwhtacoooonfjvglomnbtbgsmemrptkex
```

Рисунок 1 - Результат работы программы с использованием случайной генерации символов методом Хартли

Программа начинается с предложения пользователю выбрать режим работы: либо случайная генерация строки, либо ручной ввод. В данном случае выбран первый вариант. Пользователь вводит длину строки, которая составляет 50 символов. Программа генерирует случайную строку из маленьких букв английского алфавита длиной 50 символов.

После этого строка переводится в двоичный вид, где каждый символ представлен 8 битами. Таким образом, длина исходной строки в битах составляет 400. Затем программа разбивает символьную строку на пары символов, что облегчает дальнейшую обработку.

Далее происходит построение частотной карты для этих пар символов, чтобы определить, насколько часто каждая пара встречается. На основе этой

частотной карты строится очередь узлов, отсортированных по частоте. Эти узлы используются для построения дерева Хаффмана, в котором узлы с меньшей частотой находятся ближе к листьям дерева, а узлы с большей частотой – ближе к корню.

После построения дерева программа создаёт коды Хаффмана для каждой пары символов. Каждой паре присваивается уникальный двоичный код. Например, для данной строки программа может назначить пару 'qx' код '01110', паре 'xr' – '01111' и так далее.

Сформированные коды используются для кодирования исходной строки. Закодированное сообщение состоит из последовательности битов, представляющих каждую пару символов в соответствии с их кодом Хаффмана. В результате получается закодированное сообщение, длина которого в данном случае составляет 118 бит.

Затем программа демонстрирует, как декодировать это сообщение обратно в исходную строку. Используя дерево Хаффмана, программа преобразует последовательность битов обратно в пары символов, восстанавливая исходную строку.

Пример №2

```
Введите «1» для случайной генерации или «2» для ручного ввода: 2
Введите свою строку: Я вижу Великую Тьму!! Тьфу! Опять капюшон сполз на
 глаза...
Длина исходной строки: 604
Исходная строка: ['Я ', 'ви', 'жу', ' В', 'ел', 'ик', 'ую', ' Т', 'ьм',
 'у!', '! ', 'Ть', 'фу', '! ', 'Оп', 'ят', 'ь ', 'ка', 'пю', 'шо', 'н ',
  'сп', 'ол', 'з ', 'на', ' г', 'ла', 'за', '..', '.']
Код: {'..': '0000', '.': '0001', '! ': '0010', 'Я ': '00110', 'ви':
 '00111', 'жу': '01000', ' В': '01001', 'ел': '01010', 'ик': '01011',
 'ую': '01100', ' Т': '01101', 'ьм': '01110', 'у!': '01111', 'Ть':
 '10000', 'фу': '10001', 'Оп': '10010', 'ят': '10011', 'ь ': '10100',
 'ка': '10101', 'пю': '10110', 'шо': '10111', 'н ': '11000', 'сп':
 '11001', 'ол': '11010', 'з ': '11011', 'на': '11100', ' г': '11101',
 'ла': '11110', 'за': '11111'}
Закодированное сообщение:
 0011000111010000100101010010110110001101011100111100101000010001001010
 0
 1010011101001010110110101111100011001110101101111100111011111011111000
 0001
Длина закодированного сообщения: 146
Декодированная строка: Я вижу Великую Тьму!! Тьфу! Опять капюшон сполз
 на глаза...
```

Рисунок 2 - Результат работы программы при ручном вводе символов

Программа начинается с предложения пользователю выбрать режим работы: случайная генерация строки или ручной ввод. В этом примере выбран ручной ввод, и пользователь вводит свою строку: "**Я вижу Великую Тьму!! Тьфу! Опять капюшон сполз на глаза...**".

Далее программа переводит введённую строку в двоичный вид, где каждый символ представлен 8 битами. Поскольку введённая строка содержит 76 символов, её двоичный эквивалент имеет длину 608 бит.

Следующим шагом программа разбивает строку на пары символов, чтобы упростить процесс кодирования. В результате получаем список пар

символов: ['Я ', 'ви', 'жу', ' В', 'ел', 'ик', 'ую', ' Т', 'ьм', 'у!', '! ', 'Ть', 'фу', '! ', 'Оп', 'ят', 'ь ', 'ка', 'пю', 'шо', 'н ', 'сп', 'ол', 'з ', 'на', ' г', 'ла', 'за', '..', '.'].

Затем программа строит частотную карту для этих пар символов, определяя, как часто каждая пара встречается. На основе этой карты строится очередь узлов, отсортированных по частоте, которые используются для построения дерева Хаффмана. Узлы с меньшей частотой размещаются ближе к листьям дерева, а с большей частотой — ближе к корню.

После этого создаются коды Хаффмана для каждой пары символов. Например, 'Я ' может получить код '00110', 'ви' — '00111' и так далее. Эти коды уникальны и соответствуют частоте появления пар символов в строке.

Программа использует эти коды для кодирования исходной строки, преобразуя каждую пару символов в соответствующий ей двоичный код. В результате получается закодированное сообщение: 00110001110100001001010100101101100011010111001111001010000100010010100101001110100101011011010111100011001110101101111001110111110111110000001, длина которого составляет 146 бит.

Затем программа декодирует это сообщение обратно в исходную строку, используя построенное дерево Хаффмана. Процесс декодирования восстанавливает пары символов из последовательности битов, возвращая их в исходную строку.

Вывод

Из результатов работы программы видно, что кодирование строки методом Хаффмана позволяет значительно сократить количество бит, необходимых для передачи исходной информации. Например, исходная строка, содержащая 76 символов и имеющая длину 608 бит в двоичном виде, была закодирована в сообщение длиной всего 146 бит, что означает сокращение в более чем в четыре раза.

Код Хаффмана позволяет эффективно сжимать данные, используя переменную длину кодов для различных символов в зависимости от их частоты встречаемости. Это позволяет сделать кодирование более эффективным для часто встречающихся символов, используя более короткие коды, в то время как редкие символы могут иметь более длинные коды.

Результаты работы программы демонстрируют эффективность и универсальность метода Хаффмана в сжатии данных, что является важным аспектом для передачи и хранения информации в современных информационных системах.

Список литературы

1. Колмогоров А. Н., "Теория информации и теория алгоритмов". Издательство МГУ, 1987.
2. Лебедев В. И., "Теория информации". Физматлит, 2002.
3. Журавлев Ю. И., Кобзарев В. Н., "Теория информации и её приложения". Советское радио, 1974.
4. "Алгоритмы и структуры данных" под редакцией А.В. Ахо, И.Х. Ульмана и М.Хопкрофта, издательство "Вильямс", 2018 год.
5. "Теория кодирования информации" Штейнберг Л.А., Гуз А.Н., издательство "БХВ-Петербург", 2013 год.
6. "Кодирование информации: Классические и современные методы" Лихтарников Л.Б., издательство "Физматлит", 2016 год.
7. "Алгоритмы. Построение и анализ" Левитин А.В., издательство "Питер", 2016 год.
8. "Сжатие данных" Раскин Н.А., Мальцева И.С., издательство "Издательский дом МГТУ им. Н.Э. Баумана", 2015 год.