Лабораторная работа № 9

СЖАТИЕ/РАСПАКОВКА ДАННЫХ НА ОСНОВЕ СТАТИСТИЧЕСКИХ МЕТОДОВ

Цель: приобретение практических навыков использования статистических методов Шеннона – Фано и Хаффмана (Shannon-Fano and Huffman coding) для сжатия/распаковки данных.

Задачи:

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов сжатия/распаковки (архивации/разархивации) данных на основе методов Шеннона – Фано и Хаффмана.

2. Разработать приложение для реализации методов Шеннона – Фано и Хаффмана.

3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

9.1. Теоретические сведения

9.1.1. Описание и основные свойства статистических методов сжатия

Мы неоднократно подчеркивали, что каждый из естественных языков обладает *избыточностью*. Среди европейских языков белорусский и русский обладают одним из самых высоких уровней избыточности. Об этом можно судить по размерам русского перевода английского текста. Обычно он примерно на 20–30% больше.

До появления уже упоминавшихся работ К. Шеннона кодирование символов алфавита при передаче сообщения по каналам связи осуществлялось одинаковым количеством битов, получаемым по формуле Хартли (см. формулу (2.2)). Позднее начали появляться способы, кодирующие символы разным числом битов в зависимости от вероятности появления их в тексте, подтверждение чему мы получили при выполнении лабораторной работы № 2. Таким образом, за счет использования для каждого значения байта кодов ASCII (символа алфавита) *кода различной длины* в соответствии с частостью (вероятностью появления этого символа в

сообщении) можно значительно уменьшить общий размер данных. Эта базовая идея лежит в основе алгоритмов статистических (вероятностных) методов сжатия: Шеннона – Фано и Хаффмана.

Статистические алгоритмы позволяют создавать более короткие коды для часто встречающихся и более длинные – для редко встречающихся символов алфавита или конкретного сообщения. В первом случае метод считается статическим статистическим, во втором – динамическим статистическим: вероятностные свойства символов подсчитываются для конкретного сообщения или потока данных.

Частота или вероятность появления того или иного символа алфавита в произвольном сообщении, лежащая в основе алгоритмов, дали название этим алгоритмам и соответствующим методам.

Иногда эти методы называют также *префиксными*.

К примеру, если имеется некоторый код, который записывается как $X_1 = A_1A_2$, и другой код – $X_2 = A_1$, то говорят, что X_2 является *префиксом* X_1. Или если $X_1 = 1010$, а $X_2 = 10101100$, то X_2 также является *префиксом* X_1.

Таким образом, использование описываемых методов предусматривает создание кодовой таблицы (подобно кодам ASCII или base64). Формально процедура сжатия (прямое преобразование) состоит в подстановке соответствующего бинарного кода вместо символа исходного алфавита и наоборот – при обратном преобразовании.

Методы относятся к классу «сжатие без потерь». Различие между двумя рассматриваемыми методами состоит лишь в особенностях формирования таблицы бинарных кодов. При формировании этой таблицы для обоих методов можно воспользоваться статистическими свойствами алфавитов, полученными при выполнении лабораторной работы № 2.

Основополагающая идея методов была предложена К. Шенноном в его известной работе [26] (полезно ознакомиться с переводной версией этой работы, точнее: Ч. 1 «Дискретные системы без шума», п. «Представление операций кодирования и декодирования» [27]).

9.1.2. Метод Шеннона – Фано

Детально метод был описан Р. Фано, который опубликовал его в виде технического отчета [29].

Код Шеннона – Фано не является оптимальным (обеспечивает минимальную избыточность) в общем смысле, хотя и дает оптимальные результаты при некоторых распределениях вероятностей. Для одного и того же распределения вероятностей можно построить, вообще говоря, несколько кодов Шеннона – Фано, и все они могут дать различные результаты.

Итак, необходимо выполнить следующие действия:

1) подсчитать вероятностные параметры символов алфавита $A = \{a_i\}$ (реализуется статическая версия алгоритма);

2) отсортировать – обычно в порядке убывания (невозрастания, т. е. могут иметь место повторяющиеся значения) вероятностей $p(a_i)$; $p(a_i)$ – вероятность появления в сжимаемом сообщении на произвольной позиции символа a_i алфавита, т. е. создать таблицу символов алфавита, на основе которого генерируется сжимаемое сообщение;

3) каждому символу отсортированного множества поставить в соответствие бинарный код, для чего это множество (таблица) символов делится на две группы таким образом, чтобы каждая из групп имела приблизительно одинаковую суммарную частоту (вероятность). Очевидно, на первом шаге такая суммарная вероятность в каждой из групп должна быть максимально близка к 0,5. Первому из полученных подмножеств устанавливается первый символ бинарного кода: 0, второй – 1 (или наоборот). Для вычисления следующих битов кодов данная процедура повторяется рекурсивно для каждого из полученных на текущем шаге подмножеств, в котором содержится больше одного символа. Получим таблицу, в которой длина кодовых комбинаций меняется от минимального ($l_{\min}$) до максимального ($l_{\max}$) значений.

Пример. Имеем алфавит $A\{a_i\}$, $i = [1, \ldots, 10]$, $N(A) = 10$ – мощность алфавита. При этом получены следующие вероятности для символов алфавита:

$p(a_1) = 0{,}05$	$p(a_6) = 0{,}12$
$p(a_2) = 0{,}10$	$p(a_7) = 0{,}05$
$p(a_3) = 0{,}03$	$p(a_8) = 0{,}10$
$p(a_4) = 0{,}20$	$p(a_9) = 0{,}15$
$p(a_5) = 0{,}15$	$p(a_{10}) = 0{,}05$

Далее рассмотрим процесс создания таблицы кодов. Отсортируем таблицу символов в порядке убывания вероятностей. Разделим ее на две части (два подмножества), как показано ниже (отделены гори-

зонтальной линией). Видно, что сумма вероятностей в обоих подмножествах одинакова и равна 0,5. Символам верхней части общей таблицы определим старший символ кода (1), нижней части – 0:

$p(a_4) = 0{,}20$	1
$p(a_5) = 0{,}15$	1
$p(a_9) = 0{,}15$	1
$p(a_6) = 0{,}12$	0
$p(a_2) = 0{,}10$	0
$p(a_8) = 0{,}10$	0
$p(a_1) = 0{,}05$	0
$p(a_7) = 0{,}05$	0
$p(a_{10}) = 0{,}05$	0
$p(a_3) = 0{,}03$	0

Далее в качестве исходного рассматриваем каждое из двух подмножеств (на текущем шаге). Выполняем рекурсивно одну и ту же процедуру. В частности, для первых трех символов таблицы (для первого подмножества) имеем после следующей итерации (ее деления на два меньших подмножества и определения следующих бинарных символов кода):

$p(a_4) = 0{,}20$	11
$p(a_5) = 0{,}15$	10
$p(a_9) = 0{,}15$	10

Последняя таблица в одном из подмножеств (первая строка) содержит только один элемент. Кодирование этого элемента закончено. Далее делим на два нижнюю часть последней таблицы и получим очередные символы соответствующих кодов:

$p(a_5) = 0{,}15$	101
$p(a_9) = 0{,}15$	100

Далее переходим к кодированию символов после первого деления исходной таблицы (начиная с символа $p(a_6)$).

Выполнив стандартные операции, получим таблицу бинарных кодов, как показано ниже:

$p(a_4) = 0{,}20$	11
$p(a_5) = 0{,}15$	101
$p(a_9) = 0{,}15$	100
$p(a_6) = 0{,}12$	011

$p(a_2) = 0{,}10$ 010
$p(a_8) = 0{,}10$ 0011
$p(a_1) = 0{,}05$ 0010
$p(a_7) = 0{,}05$ 0001
$p(a_{10}) = 0{,}05$ 00000
$p(a_3) = 0{,}03$ 00001

Или после расположения символов в порядке возрастания индексов символов алфавита:

$p(a_1) = 0010$ $p(a_6) = 011$
$p(a_2) = 010$ $p(a_7) = 0001$
$p(a_3) = 00001$ $p(a_8) = 0011$
$p(a_4) = 11$ $p(a_9) = 100$
$p(a_5) = 101$ $p(a_{10}) = 00000$

Как видим, для рассмотренного примера получена минимальная длина кода, равная 2 битам ($l_{min} = 2$), и максимальная длина, равная 5 битам ($l_{max} = 5$). Действительно, символам с б*о*льшими вероятностями соответствуют коды меньшей длины ($l_{min} = 2$) и наоборот ($l_{max} = 5$). Замечаем, что выполнено основное требование: все кодовые комбинации разные. И соблюдено «требование префикса»: ни одна из кодовых комбинаций меньшей длины не является началом кодовой комбинации большей длины.

Именно последняя таблица используется в неизменном виде (речь о кодах) в процессах прямого и обратного преобразований.

Алгоритм прямого преобразования: необходимо выполнить одну операцию: заменить символы входного сообщения соответствующими бинарными кодами.

Алгоритм обратного преобразования: на входе – сообщение в виде бинарной последовательности.

Шаг 1. Анализируются l_{min} начальных бинарных символов: осуществляется поиск в таблице соответствующего совпадения. Если такое будет найдено, то на выходе будет символ исходного алфавита с совпадающим кодом. После этого процедура повторяется, т. е. анализируются очередные l_{min} символов. Если не найдено в таблице совпадения, переходим к шагу 2.

Шаг 2. Длина анализируемой последовательности увеличивается на 1 бит: $l_{min} + 1$. Осуществляется поиск совпадающей бинарной комбинации такой же длины в таблице. Если такая комбинация существует, на выходе распаковщика формируется соответст-

вующий символ исходного алфавита, если нет – длина анализируемой последовательности увеличивается еще на один бит, и т. д.

По различным причинам при анализе очередной последовательности длиной l_{max} совпадение в таблице может быть не найдено. Для нейтрализации подобных коллизий архиваторы содержат средства контроля ошибок с помощью корректирующих кодов.

Пример. *Прямое преобразование*. Итак, требуется сжать сообщение X = «$a_1a_1a_9a_9a_9a_5a_1a_5a_5a_1$» (10 символов). Заменяя символы соответствующими им кодами из таблицы, на выходе получим: X_n = 00100010100100100101001010110100 10.

Как видим, общая длина (объем после сжатия $V_{пс}$) сообщения составляет 34 бита. Если бы каждый символ сообщения X заменялся некоторым кодом, подобным ASCII, то длина его составила бы 80 битов (10 · 8).

Обратное преобразование. На входе имеем бинарную последовательность Y_n = 00100010100100100101001010110100 10.

Шаг 1: анализируются начальные 2 (l_{min} = 2) символа этой последовательности: 00. Совпадающих комбинаций в таблице нет.

Шаг 2: длину анализируемой последовательности увеличиваем на один бит: 001. Совпадение не найдено.

Шаг 3: анализируем 4-битовую комбинацию: 0010. Этой комбинации в таблице соответствует символ исходного алфавита «a_1». На выходе распаковщика будет именно этот символ.

Анализируется очередные 2 символа ($l_{мин}$ = 2): 00 (шаг 1), и т. д.

Теперь предположим, что во входном сообщении изменился только один символ (первый):

$$Y_n = \mathbf{1}0100010100100100101001010110100 10.$$

А распаковщик не содержит средств для ее обнаружения.

В таком случае первой совпадающей комбинацией будет 101: «a_5». Следующей – «a_7», за ней – «a_2», и т. д.

Следуя логике рассуждений, нетрудно подсчитать коэффициент компрессии для нашего примера (см. формулы на с. 76):

$$R_1 = 34 / 80 \text{ либо } R_2 = (80 - 34) / 80 = 46 / 80.$$

9.1.3. Метод Хаффмана

Метод основан на алгоритме *оптимального префиксного кодирования* алфавита: исходный алгоритм Хаффмана является

оптимальным для посимвольного кодирования с известным входным распределением вероятностей, т. е. для отдельного кодирования несвязанных символов в таком потоке данных [30]. Отличается от метода Шеннона – Фано лишь в части кодирования символов исходного алфавита.

В данном случае бинарные коды создаются на основе дерева, ветви которого обозначаются бинарными символами.

Бинарным кодом символа исходного алфавита будет последовательность обозначений ветвей дерева от корня до листа, соответствующего этому символу. В основе бинарного кода лежит следующее положение.

Лемма. **Для любого заданного алфавита (источника) с $N > 2$ символами существует оптимальный двоичный код, в котором два наименее вероятных символа (слова) имеют одну и ту же длину и отличаются лишь последним битом [31].**

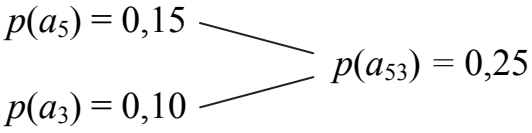
Построение дерева начинается с сортирования символов исходного алфавита в порядке убывания (невозрастания).

Далее выбираются два символа (a_i, a_j) с наименьшими вероятностями ($p(a_i)$, $p(a_j)$) и объединяются в узел. Ветви этого узла обозначаются «1» и «0». Этот узел рассматривается далее как новый, виртуальный символ (a_{ij}), которому будет соответствовать вероятность $p(a_{ij}) = p(a_i) + p(a_j)$. Такой виртуальный символ будет рассматриваться далее наравне с остальными символами исходного алфавита. Два его потомка из дальнейшего рассмотрения исключаются. Создаются новые узлы дерева по тому же принципу. Корень дерева образуют два символа с наибольшими вероятностями.

Пример. Пусть имеется алфавит из пяти символов, который в отсортированном виде выглядит так:

$p(a_4) = 0{,}35$
$p(a_2) = 0{,}20$
$p(a_1) = 0{,}20$
$p(a_5) = 0{,}15$
$p(a_3) = 0{,}10$

Строим дерево. Два нижних символа создают первый узел этого дерева, который обозначаем a_{53}. Ему соответствует вероятность $p(a_{53}) = 0{,}25$.



Верхней ветви этого узла будет соответствовать «1», нижней – «0».

Далее рассматриваем алфавит, состоящий из следующих символов: $p(a_4) = 0{,}35$, $p(a_{53}) = 0{,}25$, $p(a_2) = 0{,}20$, $p(a_1) = 0{,}20$, $p(a_5) = 0{,}15$.

Строго говоря, новая последовательность опять должна быть отсортирована. Далее снова нужно объединить два символа с наименьшими вероятностями, и т. д. С другой стороны, при построении дерева можно придерживаться следующего правила: на одном уровне иерархии дерева в узлы объединяются символы с примерно одинаковыми вероятностями; при обозначении ветвей двоичная единица присваивается символу с большей вероятностью.

Ниже представлена таблица (один из вариантов) соответствий между символами исходного алфавита и их бинарными кодами:

a_1	01
a_2	00
a_3	100
a_4	11
a_5	101

Как видно, для одного и того же алфавита по обоим рассмотренным статистическим методам могут быть созданы разные таблицы, т. е. одним и тем же символам могут соответствовать разные кодовые комбинации. Наилучшей (оптимальной или близкой к оптимальной) является та таблица, которой соответствует минимальное значение интегрального коэффициента C:

$$C = \sum p(a_i) \cdot l_i,$$

где $p(a_i)$ и l_i – соответственно вероятность появления символа и длина бинарного кода для этого символа.

Для данных из последнего примера имеем:

$$C = 0{,}20 \cdot 2 + 0{,}20 \cdot 2 + 0{,}10 \cdot 3 + 0{,}35 \cdot 2 + 0{,}15 \cdot 3 = 2{,}1 \text{ бита}.$$

Это означает, что для выполненной кодировки одному символу исходного алфавита в среднем соответствует 2,1 бита.

Д. Хаффман составил дерево для алфавита английского языка, которое является оптимальным. В таблице ниже представлены эти коды.

Бинарные коды английского алфавита в соответствии с деревом Хаффмана

Символ алфавита	Бинарный код
E	100
T	001
A	1111
O	1110
N	1100
R	1011
I	1010
S	0110
H	0101
D	11011
L	01111
F	01001
C	01000
M	00011
U	00010
G	00001
Y	00000
P	110101
W	011101
B	011100
V	1101001
K	110100011
X	110100001
J	110100000
Q	1101000101
Z	1101000100

Как видно, в текстах на английском языке наиболее часто встречается буква «e» (ей соответствует вероятность 0,127).

Хороший пример программной реализации рассмотренных методов можно найти в Интернет-источнике [32].

9.2. Практическое задание

1. Разработать авторское приложение в соответствии с целью лабораторной работы.

2. С помощью приложения выполнить прямое и обратное преобразования сообщения, состоящего из собственных имени и фамилии.

Можно использовать любой из известных методов сортировки символов массива. Метод кодировки (Шеннона – Фано, Хаффмана) использовать по указанию преподавателя.

При этом таблица отсортированных символов строится:

а) на основе данных, полученных в лабораторной работе № 2;

б) динамически, на основе анализа сжимаемого сообщения.

3. Определить эффективность (в сравнении с кодами ASCII) сжатия сообщения.

4. Результаты оформить в виде отчета по установленным правилам.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Что такое бинарное дерево, чем характеризуется его структура?

2. Какие коды называются префиксными?

3. Как, на Ваш взгляд, при увеличении мощности алфавита меняется его избыточность (или не меняется)?

4. В чем состоит суть кодирования по методам Шеннона – Фано и Хаффмана?

5. Построить бинарное дерево для метода Шеннона – Фано (сообщение – собственная фамилия).

6. В каких известных Вам архиваторах используются префиксные методы?

7. Сравнить эффективность кодирования по двум анализируемым методам. Показать на конкретном примере.