Государственный комитет информатизации и связи Украины
Одесская национальная академия связи им. А. С. Попова
Кафедра документальной электросвязи

СЖАТИЕ ДАННЫХ БЕЗ ПОТЕРЬ

Методическое пособие
к практическому занятию и лабораторной работе
по курсу
«СИСТЕМЫ ДОКУМЕНТАЛЬНОЙ ЭЛЕКТРОСВЯЗИ»

Одесса 2006

1 ПРАКТИЧЕСКОЕ ЗАНЯТИЕ И ЛАБОРАТОРНАЯ РАБОТА №1

СРАВНЕНИЕ АЛГОРИТМОВ СЖАТИЯ БЕЗ ПОТЕРЬ

1.1 Ключевые положения

Подавляющее большинство современных форматов записи данных представлены в виде, удобном для быстрого манипулирования и удобного прочтения пользователями. При этом данные занимают объем больший, чем это действительно требуется для их хранения. Алгоритмы, которые устраняют избыточность записи данных, называются алгоритмами сжатия данных, или алгоритмами архивации.

Технические характеристики процессов сжатия

Целью процесса сжатия является получение более компактного выходного потока из изначально некомпактного входного потока. Основными техническими характеристиками процессов сжатия и результатов их работы являются:

- коэффициент сжатия $K_{сж}$, или отношение объемов исходного и результирующего потоков;
- скорость сжатия - время, затрачиваемое на сжатие входного потока, до получения из него эквивалентного выходного потока;
- качество сжатия - величина, показывающая, на сколько сильно упакован выходной поток, при помощи применения к нему повторного сжатия по этому же или иному алгоритму.

Все способы сжатия можно разделить на две категории: *обратимое* и *необратимое*.

Под *необратимым* сжатием (архивация с потерями) подразумевают такое преобразование входного потока данных, при котором выходной поток представляет достаточно похожий на входной поток объект, однако отличается от него объемом.

Степень сходства входного и выходного потоков определяется степенью соответствия некоторых свойств объекта (т.е. сжатой и несжатой информации в соответствии с некоторым определенным форматом данных), представляемого данным потоком информации.

Обратимое сжатие (архивация без потерь) всегда приводит к снижению объема выходного потока информации без изменения его информативности, т.е. - без потери информационной структуры. Из выходного потока, при помощи восстанавливающего алгоритма, можно получить входной. Процесс восстановления называется декомпрессией или распаковкой.

Для того чтобы корректнее оценивать степень сжатия, нужно ввести понятие класса изображений. Под классом будет пониматься некая совокупность изображений, применение к которым алгоритма архивации дает качественно одинаковые результаты. Например, для одного класса алгоритм дает очень высокую степень сжатия, для другого – почти не сжимает, для третьего – увеличивает файл в размере. (Известно, что многие алгоритмы в худшем случае увеличивают файл). Рассмотрим следующие примеры неформального определения классов изображений:

Класс 1 Изображения с небольшим количеством цветов (4-16) и большими областями, заполненными одним цветом. Плавные переходы цветов отсутствуют. Примеры: деловая графика – гистограммы, диаграммы, графики и т.п.

Класс 2 Изображения, с плавными переходами цветов, построенные на компьютере. Примеры: графика презентаций, эскизные модели в САПР, изображения, построенные по методу Гуро.

Класс 3 Фотореалистичные изображения. Пример: отсканированные фотографии.

Класс 4 Фотореалистичные изображения с наложением деловой графики. Пример: реклама.

Алгоритм RLE

Данный алгоритм необычайно прост в реализации. Групповое кодирование – от английского Run Length Encoding (RLE) – один из самых старых и самых простых алгоритмов архивации. Сжатие в RLE происходит за счет того, что в исходном потоке встречаются цепочки одинаковых байт. Замена их на пары <счетчик повторений, значение> уменьшает избыточность данных. Ситуация, когда файл увеличивается, для этого простого алгоритма не так уж редка.

Рассмотрим пример архивации. Пусть 44 44 44 11 11 11 11 11 01 33 97 22 22 (13 байт) – исходная последовательность. В сжатой последовательности первый байт указывает, сколько раз нужно повторить следующий байт, поэтому для передачи первых трех байт будет использоваться 2 байта – 03 и 44, вместо пяти последующих – 05 и 11. Если далее находятся неповторяющиеся данные, то первый байт равен 00, а затем идет счетчик, показывающий сколько за ним следует неповторяющихся данных и сами данные – 00 03 01 33 97. Сжатая последовательность будет иметь вид: 03 44 05 11 00 03 01 03 97 02 22 (11 байт). Коэффициент сжатия при этом составит $K_{сж}$ = 13/11= 1,18.

Данные методы, как правило, достаточно эффективны для сжатия растровых графических изображений (BMP, PCX, TIFF, GIF), т.к. последние

содержат достаточно много длинных серий повторяющихся последовательностей байтов. Недостатком метода RLE является достаточно низкая степень сжатия.

Алгоритм LZW

Название алгоритм получил по первым буквам фамилий его разработчиков – Lempel, Ziv и Welch. Сжатие в нем, в отличие от RLE, осуществляется уже за счет *одинаковых цепочек* байт. Существует довольно большое семейство LZ-подобных алгоритмов, различающихся методом поиска повторяющихся цепочек.

Первой вещью, которую мы делаем при LZW-сжатии, является инициализация цепочки символов. Чтобы сделать это, нам необходимо выбрать код размера (количество бит) и знать, сколько возможных значений могут принимать наши символы. Давайте положим выходной код размером 12 бит, что означает возможность запоминания 4096 элементов в таблице строк. Давайте также предположим, что мы имеем 256 возможных различных символов (например, ASCII-таблица или изображение с глубиной цвета 8 бит). Чтобы инициализировать таблицу, мы установим соответствие кода <0> символу 0, кода <1> символу 1, и т.д., до кода <255> и символа 255. Для кода очистки и кода конца информации зарезервированы значения <256> и <257>. Под коды для строк нам остаются значения от <258> до <4095>.

Процесс сжатия выглядит достаточно просто. Мы считываем последовательно символы входного потока и проверяем, есть ли в созданной нами таблице строк такая строка. Если строка есть, то мы считываем следующий символ, а если строки нет, то мы заносим в поток код для предыдущей найденной строки, заносим строку в таблицу и начинаем поиск снова. Добавляемые строки записываются в таблицу последовательно, при этом индекс строки в таблице становится ее кодом.

Пусть мы сжимаем последовательность 45 55 55 151 55 55 55. Сначала мы поместим в выходной поток код очистки <256>, потом добавим к изначально пустой строке 45 и проверим, есть ли строка <45> в таблице. Поскольку мы при инициализации занесли в таблицу все строки из одного символа, то строка <45> есть в таблице. Далее мы читаем следующий символ 55 из входного потока и проверяем, есть ли строка 45 55 в таблице. Такой строки в таблице пока нет. Мы заносим в таблицу строку 45 55 (с первым свободным кодом <258>) и записываем в поток код <45> и т.д. (см. табл. 1.1)

Последовательность кодов для данного примера, попадающих в выходной поток: <256>, <45>, <55>, <55>, <151>, <259>, <55> и код завершения информации <257>.

Таблица 1.1 – Сжатие последовательности по алгоритму LZW

Строка	Наличие в таблице строк	Добавление в таблицу строк	Выходной поток
45	есть (номер <45>)		
45 55	нет	<258> 45 55	<45>
55 55	нет	<259> 55 55	<55>
55 151	нет	<260> 55 151	<55>
151 55	нет	<261> 151 55	<151>
55 55	есть (номер <259>)		
55 55 55	нет	<262> 55 55 55	<259>

Особенность LZW заключается в том, что для декомпрессии нам не надо сохранять таблицу строк в файл для распаковки. Алгоритм построен таким образом, что мы в состоянии восстановить таблицу строк, пользуясь только потоком кодов.

Замечание. Как вы могли заметить, записываемые в поток коды постепенно возрастают. До тех пор, пока в таблице не появится, например, в первый раз код 512, все коды будут меньше 512. Кроме того, при компрессии и при декомпрессии коды в таблице добавляются при обработке одного и того же символа, т.е. это происходит “синхронно”. Мы можем воспользоваться этим свойством алгоритма для того, чтобы повысить степень компрессии. Пока в таблицу не добавлен 512 символ, мы будем писать в выходной битовый поток коды из 9 бит, а сразу при добавлении 512 – коды из 10 бит. Соответственно декомпрессор также должен будет воспринимать все коды входного потока 9-битными до момента добавления в таблицу кода 512, после чего будет воспринимать все входные коды как 10-битные. Аналогично мы будем поступать при добавлении в таблицу кодов 1024 и 2048. Данный прием позволяет примерно на 15% поднять степень компрессии:

Классический алгоритм Хаффмана

Один из классических алгоритмов, известных с 60-х годов. Использует только частоту появления одинаковых байт в изображении. Сопоставляет символам входного потока, которые встречаются большее число раз, цепочку бит меньшей длины. И, напротив, встречающимся редко – цепочку большей длины.

Определение. Рассмотрим соответствие между буквами алфавита Y и некоторыми словами алфавита W:

$$a_1 - B_1, \qquad a_2 - B_2, \qquad \ldots \qquad a_r - B_r$$

Это соответствие называют схемой и обозначают через S . Оно определяет кодирование следующим образом: каждому слову $A = a_{i1}a_{i2}\ldots a_{in}$ из $S'(W)=S(W)$ ставится в соответствие слово $B = B_{i1}B_{i2}\ldots B_{in}$, называемое кодом слова А. Слова B_1

... B_r называются элементарными кодами. Данный вид кодирования называют алфавитным кодированием.

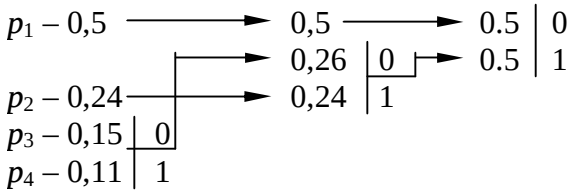
Алгоритм построения схемы S можно представить следующим образом:

Шаг 1 Упорядочиваем все буквы входного алфавита в порядке убывания вероятности. Считаем все соответствующие слова B_i из алфавита W ={0,1} пустыми.

Шаг 2 Объединяем два символа a_{ir-1} и a_{ir} с наименьшими вероятностями p_{ir-1} и p_{ir} в псевдосимвол $a'\{a_{ir-1}\ a_{ir}\}$ с вероятностью $p_{ir-1}+p_{ir}$. Дописываем 0 в начало слова B_{ir-1} ($B_{ir-1} = 0\ B_{ir-1}$), и 1 в начало слова B_{ir} ($B_{ir}=1B_{ir}$).

Шаг 3 Удаляем из списка упорядоченных символов a_{ir-1} и a_{ir}, заносим туда псевдосимвол $a'\{a_{ir-1}\ a_{ir}\}$. Проводим шаг 2, добавляя при необходимости 1 или ноль для всех слов B_i, соответствующих псевдосимволам, до тех пор, пока в списке не останется 1 псевдосимвол.

Пример: Пусть у нас есть 4 символа в алфавите $Y =\{a_1,..., a_4\}$ (r=4), p_1=0.5, p_2=0.24, p_3=0.15, p_4=0.11. Тогда процесс построения схемы можно представить так:



Производя действия, соответствующие 2-му шагу, мы получаем псевдосимвол с вероятностью 0.26 (и приписываем 0 и 1 соответствующим словам). Повторяя же эти действия для измененного списка, мы получаем псевдосимвол с вероятностью 0.5. И, наконец, на последнем этапе мы получаем суммарную вероятность 1.

Для того, чтобы восстановить кодовые слова, нам надо пройти по стрелкам от начальных символов к концу получившегося бинарного дерева. Так, для символа с вероятностью p_4, получим B_4=101, для p_3 получим B_3=100, для p_2 получим B_2=11, для p_1 получим B_1=0. Что означает схему:

$$a_1 - 0 \qquad a_2 - 11 \qquad a_3 - 100 \qquad a_4 - 101$$

Эта схема представляет собой префиксный код, являющийся кодом Хаффмана. Самый часто встречающийся в потоке символ a_1 мы будем кодировать самым коротким словом 0, а самый редко встречающийся a_4 длинным словом 101.

Для последовательности из 100 символов (будем считать, что длина 1 символа равна 1 байту, т.е. 800 бит), в которой символ a_1 встретится 50 раз, символ a_2 – 24 раза, символ a_3 – 15 раз, а символ a_4 – 11 раз, данный код позволит

получить последовательность из 176 бит (50*1+24*2+15*3+11*3). Т.е. коэффициент сжатия составит $K_{сж}$ = 800/176 = 4,55.

Как стало понятно из изложенного выше, классический алгоритм Хаффмана требует записи в файл таблицы соответствия кодируемых символов и кодирующих цепочек.

На практике используются его разновидности. Так, в некоторых случаях резонно либо использовать постоянную таблицу, либо строить ее “адаптивно”, т.е. в процессе архивации/разархивации. Эти приемы избавляют нас от двух проходов по изображению и необходимости хранения таблицы вместе с файлом. Кодирование с фиксированной таблицей применяется в качестве последнего этапа архивации в JPEG и в алгоритме, разработанном третьей группой по стандартизации Международного Консультационного Комитета по Телеграфии и Телефонии (CCITT Group 3) – модифицированном коде Хаффмана.

1.2 Задание к практическому занятию

1 По разделу «Сжатие данных» данного пособия изучить принципы построения сжатых последовательностей с помощью каждого из приведенных алгоритмов.

2 По последним 2-м цифрам номера студенческого билета из таблицы 1.2 выбрать информационные последовательности для сжатия. Для алгоритма LZW использовать последовательность, повторенную дважды.

Для классического кода Хаффмана сначала необходимо рассчитать вероятности появления байт в последовательности.

Таблица 1.2 – Информационные последовательности для сжатия

0	11 44 11 44 44 44 44
1	151 56 78 78 78 78
2	56 56 23 23 23 02 02
3	02 02 02 44 44 11 11
4	23 23 23 11 11 78 89
5	02 56 11 11 44 44 44
6	44 44 11 11 11 44 56
7	20 20 11 11 11 89 02
8	44 44 44 11 78 89 89
9	151 151 151 02 02

3 Рассчитать коэффициенты сжатия для полученных последовательностей (при расчете коэффициента для алгоритма LZW следует помнить, что входной поток был 8-битным, а выходной – 9-битным). Сделать выводы о полученных результатах. Сравнить алгоритмы по степени сжатия.

1.3 Задание к лабораторной работе

Целью лабораторной работы является сравнение алгоритмов сжатия данных без потерь, примененных к различным классам изображений. В процессе выполнения лабораторной работы необходимо:

1. Определить класс изображений, представленных в папке «Сжатие без потерь».

2 Для каждого класса в отдельности провести перекодировку изображений в формат tiff с использованием сжатия по алгоритму LZW, смешанному алгоритму Хаффмана и RLE, а также модифицированному алгоритму Хаффмана.

3 Рассчитать коэффициент сжатия для каждого алгоритма и класса. Результаты вычислений занести в таблицу 1.2.

Таблица 1.2 – Сравнение алгоритмов сжатия

Имя файла	**Класс изображений**	**Алгоритм сжатия**	**Размер исходного файла, кб**	**Размер сжатого файла, кб**	**Коэффициент сжатия, $К_{сж}$**	**Потери (есть/ нет)**
	1	LZW				
		Хаффман и RLE				
		CCITT Group 3				
	2	LZW				
		Хаффман и RLE				
		CCITT Group 3				
	3	LZW				
		Хаффман и RLE				
		CCITT Group 3				

4 По результатам вычислений сделать выводы о пригодности алгоритмов сжатия для предложенных классов изображений

1.4 Содержание протокола

Протокол лабораторной работы должен содержать: тему и цель лабораторной работы, расчеты по п. 1.2, заполненную таблицу 1.2, а также выводы о полученных результатах

1.5 Контрольные вопросы

1 Назовите основные технические характеристики процессов сжатия.

2 В чем разница между алгоритмами с потерей информации и без потери информации?

3 На какой класс изображений ориентирован алгоритм RLE?

4 Приведите два примера “плохих” изображений для алгоритма RLE, для которых файл максимально увеличится в размере.

5 За счет чего происходит сжатие в алгоритме LZW?

6 Приведите пример “плохого” изображения для алгоритма LZW, для которого файл максимально увеличится в размере.

7 На какой класс изображений ориентирован классический алгоритм Хаффмана?

8 Приведите пример “плохого” изображения для классического алгоритма Хаффмана.

9 Сравните алгоритмы сжатия изображений без потерь.

10 В каких случаях алгоритмы сжатия без потерь сжимают изображение с потерями?