Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Ярославский государственный университет им. П. Г. Демидова
Кафедра компьютерных сетей

Методы сжатия

Методические указания

Рекомендовано
Научно-методическим советом университета
для студентов, обучающихся по специальности
Прикладная математика и информатика

Ярославль 2009

УДК 519.2
ББК В 182я73
М 54

Рекомендовано
Редакционно-издательским советом университета
в качестве учебного издания. План 2009 года

Рецензент
кафедра компьютерных сетей
Ярославского государственного университета им. П. Г. Демидова

Составитель М. В. Краснов

Методы сжатия: метод. указания / сост. М. В. Краснов; Яросл. гос. ун-т им. П. Г. Демидова. – Ярославль : ЯрГУ, 2009. – 44 с.

М 54

Основное использование вычислительной техники связано с хранением и передачей информации, вследствие чего возникает задача об экономном методе ее записи. Методические указания содержат описание некоторых математических понятий и приемов, используемых при решении этой задачи.

Предназначены для студентов, обучающихся по специальности 010501 Прикладная математика и информатика (дисциплина «Теория информации и кодирования», блок СД), очной формы обучения.

УДК 519.2
ББК В 182я73

Введение

В настоящее время электронная вычислительная техника применяется во многих сферах человеческой деятельности. Основное использование вычислительной техники связано с хранением и организацией доступа к информации. При этом возникает задача о сжатии данных. Цель сжатия данных – обеспечить компактное представление данных, вырабатываемых источником, для их более экономного хранения и передачи по каналам связи.

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

Обратимое сжатие, или сжатие без потерь, приводит к снижению объема выходного потока информации без изменения его информативности, т. е. без потери информационной структуры. Из выходного потока, полученного после выполнения алгоритма обратимого сжатия, при помощи декодирования получается поток, в точности совпадающий с исходным.

Под необратимым сжатием, или сжатием с потерями, подразумевают такое преобразование входного потока данных, при котором выходной поток представляет из себя объект, с некоторой точки зрения достаточно похожий по внешним характеристикам на входной поток, однако отличается от него объемом.

Таким образом, сжатие с потерями состоит из двух этапов:

а) выделение сохраняемой части информации с помощью модели, зависящей от цели сжатия;

б) сжатие без потерь.

Такие подходы и алгоритмы используются для сжатия, например данных растровых графических файлов, где на выходе нужно представить графическую картинку, очень похожую на оригинал, но не обязательно являющуюся его точной копией.

Заметим, что сжатию могут подвергаться не только сами исходные данные, но и какие-либо преобразования над ними.

Методы сжатия без потерь

Рассмотрим методы сжатия без потерь на основе статистических алгоритмов (кодирование по Хаффману и арифметическое кодирование), а также на основе алгоритма RLE.

В основе этих методов сжатия лежит идея: «Если представлять часто используемые элементы короткими кодами, а редко используемые – длинными кодами, то для хранения блока данных требуется меньший объем памяти, чем если бы все элементы представлять кодами одинаковой длины».

Однако хочется заметить, что ни один компрессор не может сжать любой файл. После обработки любым компрессором размер части файлов уменьшится, а оставшейся части – увеличится или останется неизменным.

Утверждение. Элемент s_i, вероятность появления которого равняется $p(s_i)$, выгоднее всего представлять $-\log_2 p(s_i)$ битами.

Если распределение вероятностей F неизменно и вероятности появления элементов независимы, то среднюю длину кодов в битах можно найти как

$$H = -\sum p(s_i)\log_2 p(s_i),$$

это значение также называют энтропией источника в заданный момент времени.

Обычно вероятность появления элемента является условной. Тогда при кодировании очередного элемента s_i распределение вероятностей F принимает одно из возможных значений F_k и соответственно $H = H_k$. Среднюю длину кодов в битах можно вычислить по формуле

$$H = -\sum_k P_k H_k = -\sum_{k,i} P_k p_k(s_i)\log p_k(s_i),$$

где P_k – вероятность того, что F принимает k-е значение, которому соответствует набор вероятностей $p_k(s_i)$ генерации всех возможных элементов s_i.

Статистические алгоритмы нуждаются в знании вероятностей появления символов, оценкой, которой может являться частота появления символов во входных данных.

С учетом этого статистические алгоритмы можно разделить на три класса:

• Неадаптивные. Они используют фиксированные, заблаговременно заданные вероятности символов. Таблица вероятностей символов не передается вместе с файлом, поскольку она известна заблаговременно.

• Полуадаптивные. Для каждого файла строится таблица частот символов, и на её основе сжимают файл. Вместе со сжатым файлом передается таблица символов. Кодирование происходит за два этапа (на первом происходит подсчет частот, а на втором – кодирование).

• Адаптивные. Они начинают работать с фиксированной начальной таблицей частот символов (обычно все символы сначала равновероятны), и в процессе работы эта таблица изменяется в зависимости от символов, которые встречаются в файле. Кодирование происходит за один проход.

Алгоритм Хаффмана

Исходными данными алгоритма являются:

• алфавит $A = \{a_1, ..., a_n\}$, состоящий из n символов;

• $p(a_i)$ – вероятность появления каждого символа алфавита в рассматриваемом сообщении.

Алгоритм Хаффмана состоит из нескольких шагов.

Шаг 1. Выстраиваем все символы текущего алфавита в порядке убывания вероятностей.

Шаг 2. Строим новый алфавит A_j, который получается из предыдущего заменой двух символов $a_{i_{r-1}}$ a_{i_r} (с наименьшими вероятностями) на псевдосимвол a', причем $p(a') = p(a_{i_{r-1}}) + p(a_{i_r})$.

Продолжая эту процедуру (шаг 1 – шаг 2), будем приходить ко все более коротким алфавитам; после $n-2$ – кратного сжатия придем к алфавиту A_{n-2}, содержавшему уже всего две буквы.

Шаг 3. Символам последнего алфавита поставим в соответствие кодовые обозначения 0 и 1.

Шаг 4. Пусть кодовые обозначения уже приписаны всем символам алфавита A_j. Символам предыдущего алфавита A_{j-1} (где A_0 исходный алфавит A) кодовые обозначения приписываются:

– если $a \in A_j$ и $a \in A_{j-1}$, то в алфавите A_{j-1} он будет иметь те же кодовые обозначения, что и в алфавите A_j;

– если элементы $a_{i_{r-1}}$, a_{i_r} принадлежат алфавиту A_{j-1} и не принадлежат алфавиту A_j (они были заменены на элемент $a' \in A_j$), то их кодовые обозначения получаются из кодового обозначения элемента a' добавлением цифр 0 и 1 в конце.

Выполняем шаг 4, пока всем элементам исходного алфавита A не будет сопоставлено кодовое обозначение. Алгоритм окончен.

Легко заметить, что кодирование некоторого алфавита по методу Хаффмана не является однозначно определенной процедурой. Например, на любом этапе построения кода можно заменить цифру 1 на цифру 0, и наоборот; при этом получатся два различных кода.

Пример. Дан алфавит $A = \{"б","е","з","м","л","ь"\}$, для каждого символа из алфавита A задана вероятность его появления в тексте.

символ	е	б	з	м	л	ь
вероятность	$\frac{4}{10}$	$\frac{1}{10}$	$\frac{2}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$

Закодировать алфавит с помощью алгоритма Хаффмана.

Решение

Процесс перехода к более коротким алфавитам можно описать следующей таблицей.

	Вероятности				
	исходный алфавит A	сжатые алфавиты			
	A	A_1	A_2	A_3	A_4
е	0,4	0,4	0,4	0,4	0,6
З	0,2	0,2	0,2	0,4	0,4
Б	0,1	0,2	0,2	0,2	
м	0,1	0,1	0,2		
Л	0,1	0,1			
Ь	0,1				

Символам последнего алфавита A_4 поставим в соответствие кодовые обозначения 0 и 1. Процесс присвоения символам исходного алфавита кодовых обозначений можно описать следующей таблицей.

вероятности и кодовые обозначения.										
сжатые алфавиты								исходный алфавит		
A_4		A_3		A_2		A_1		A_0		
0,6	**1**	0,4	**0**	0,4	**0**	0,4	**0**	0,4	**0**	*е*
0,4	**0**	0,4	**11**	0,2	**10**	0,2	**10**	0,2	**10**	*з*
		0,2	**10**	0,2	**111**	0,2	**111**	0,1	**1111**	*б*
				0,2	**110**	0,1	**1101**	0,1	**1110**	*м*
						0,1	**1100**	0,1	**1101**	*л*
								0,1	**1100**	*ь*

В результате кодирования строки «еьбмззееел» получаем последовательность длинной 4*1+2*2+4*4=24 бита.

Пример окончен.

Динамический алгоритм Хаффмана

Исходными данными алгоритма являются:

• алфавит $A = \{a_1, ..., a_n\}$, состоящий из n символов;

• строка $B = b_1 b_2 b_3 \ldots b_s$, которую надо закодировать и элементы которой принадлежат алфавиту A.

Основная идея динамического кодирования заключается в том, что моделирование источника (таблица вероятности) видоизменяется после обработки очередного считываемого символа.

Рассмотрим вариант динамического алгоритма Хаффмана, который очень легко реализуется, и хотя он не является очень эффективным, хорошо иллюстрирует идею динамического кодирования. Основной структурой рассматриваемого алгоритма будем считать таблицу размера $n \times 3$.

Алгоритм кодирования состоит из нескольких шагов.

Шаг 1. Инициализация.

1. Строим множество из n кодов переменной длины на основе какого-нибудь дерева Хаффмана и случайным образом (например, по возрастанию) присвоим эти коды символам алфавита.

2. Строим таблицу размера $n \times 3$ (столбцы которой хранят: наименование символа, частоту символа в тексте, код символа).

Шаг 2. Работа алгоритма (шаг два выполняется до конца строки, которую надо закодировать).

1. Считываем очередной символ b_j.

2. В выходной файл записываем код b_j.

3. Изменяем счетчик частот во втором столбце.

4. Упорядочиваем таблицу по второму столбцу, но перемещаем только первый и второй столбцы. Коды в третьем столбце не меняются.

Алгоритм окончен.

Легко заметить, что декодирование сжатых данных осуществляется путем замены кода на соответствующий символ. При декодировании надо выполнять те же действия по построению основной таблицы размера $n \times 3$, что и при кодировании.

Пример. Дан алфавит $A = \{"з", "о", "л", "т"\}$ и строка $B = "золото"$.

Закодировать динамическим методом Хаффмана строку B.

Решение.

Процесс кодирования можно описать следующей таблицей.

Инициализация		
Символ	Счетчик	Код символа
з	0	0
л	0	10
о	0	111
т	0	110

Считываем символ «з»		
Символ	Счетчик	Код символа
з	1	0
л	0	10
о	0	111
т	0	110
выводим в файл **0**		

Считываем символ «о»		
Символ	Счетчик	Код символа
з	1	0
о	1	10
л	0	111
т	0	110
выводим в файл **10**		

Считываем символ «л»		
Символ	Счетчик	Код символа
з	1	0
о	1	10
л	1	111
т	0	110
выводим в файл **111**		

Считываем символ «о»		
Символ	Счетчик	Код символа
о	2	0
з	1	10
л	1	111
т	0	110
выводим в файл **10**		

Считываем символ «т»		
Символ	Счетчик	Код символа
о	2	0
з	1	10
л	1	111
т	1	110
выводим в файл **110**		

Считываем символ «о»		
Символ	Счетчик	Код символа
о	3	0
з	1	10
л	1	111
т	1	110
выводим в файл **0**		

Пример окончен.

Адаптированное арифметическое сжатие

Исходными данными алгоритма являются:

• алфавит $A=\{a_1,...,a_n\}$, состоящий из n символов;

• строка $B=b_1b_2b_3\ldots b_s$, которую надо закодировать и элементы которой принадлежат алфавиту A.

Арифметическое кодирование основано на идее представления кодируемого текста в виде дроби. Рассмотрим процесс кодирования (построим дробь на интервале $[0,1)$).

Алгоритм компрессии состоит из нескольких шагов.

Шаг 1. Инициализация.

1. Введем вспомогательные переменные: нижняя_граница, верхняя_граница, интервал, n, а также для каждого элемента алфавита a_i введем счетчик n_{a_i}.

2. Перед основным шагом алгоритма будем считать, что

• нижняя_граница=0,0;

• верхняя_граница=интервал=1,0;

• $n=|A|$;

• $n_{a_i}=1$ для любого символа $a_i\in A$.

Шаг 2. Работа алгоритма (шаг два выполняется до конца строки, которую надо закодировать).

1. Построить таблицу распределения вероятностей

символ	вероятность	интервал	верхняя граница символа	нижняя граница символа
a_1	$\frac{n_{a_1}}{n}$	$\left[0,\frac{n_{a_1}}{n}\right)$	$\frac{n_{a_1}}{n}$	0
…	…	…	…	…
a_i	$\frac{n_{a_i}}{n}$	$\left[\frac{n_{a_{i-1}}}{n},\frac{n_{a_{i-1}}}{n}+\frac{n_{a_i}}{n}\right)$	$\frac{n_{a_{i-1}}}{n}+\frac{n_{a_i}}{n}$	$\frac{n_{a_{i-1}}}{n}$

2. Считываем очередной символ b_i.

3. Интервал = верхняя_граница – нижняя_граница.

4. Верхняя_граница = нижняя_граница + интервал * верхняя граница для очередного символа.

5. Нижняя_граница = нижняя_граница + интервал * нижняя граница для очередного символа.

$n_{b_i} = n_{b_i} + 1;\ \ n = n + 1$

Шаг 3. Выдать любое число, принадлежащее интервалу [нижняя_граница верхняя_граница).

Алгоритм окончен.

Пример. Дан алфавит $A = \{"к","р","с","а","\$"\}$ и строка $B = "краска\$"$. Символ "$" будет обозначать конец кодируемой строки.

Произвести компрессию адаптированным арифметическим методом.

Решение.

Располагать символы в таблице вероятностей будем в алфавитном порядке.

Процесс кодирования можно описать следующей таблицей.

символ	таблица вероятностей					интервал	нижняя граница	верхняя граница
	«а»	«к»	«р»	«с»	«$»		0	1
к	$\left[0,\frac{1}{5}\right)$	$\left[\frac{1}{5},\frac{2}{5}\right)$	$\left[\frac{2}{5},\frac{3}{5}\right)$	$\left[\frac{3}{5},\frac{4}{5}\right)$	$\left[\frac{4}{5},1\right)$	1	$\frac{1}{5}$	$\frac{2}{5}$
р	$\left[0,\frac{1}{6}\right)$	$\left[\frac{1}{6},\frac{3}{6}\right)$	$\left[\frac{3}{6},\frac{4}{6}\right)$	$\left[\frac{4}{6},\frac{5}{6}\right)$	$\left[\frac{5}{6},1\right)$	$\frac{1}{5}$	$\frac{9}{30}$	$\frac{10}{30}$
а	$\left[0,\frac{1}{7}\right)$	$\left[\frac{1}{7},\frac{3}{7}\right)$	$\left[\frac{3}{7},\frac{5}{7}\right)$	$\left[\frac{5}{7},\frac{6}{7}\right)$	$\left[\frac{6}{7},1\right)$	$\frac{1}{30}$	$\frac{9}{30}$	$\frac{64}{210}$
с	$\left[0,\frac{2}{8}\right)$	$\left[\frac{2}{8},\frac{4}{8}\right)$	$\left[\frac{4}{8},\frac{6}{8}\right)$	$\left[\frac{6}{8},\frac{7}{8}\right)$	$\left[\frac{7}{8},1\right)$	$\frac{1}{210}$	$\frac{510}{1680}$	$\frac{511}{1680}$
к	$\left[0,\frac{2}{9}\right)$	$\left[\frac{2}{9},\frac{4}{9}\right)$	$\left[\frac{4}{9},\frac{6}{9}\right)$	$\left[\frac{6}{9},\frac{8}{9}\right)$	$\left[\frac{8}{9},1\right)$	$\frac{1}{1680}$	$\frac{4592}{15120}$	$\frac{4594}{15120}$
а	$\left[0,\frac{2}{10}\right)$	$\left[\frac{2}{10},\frac{5}{10}\right)$	$\left[\frac{5}{10},\frac{7}{10}\right)$	$\left[\frac{7}{10},\frac{9}{10}\right)$	$\left[\frac{9}{10},1\right)$	$\frac{2}{15120}$	$\frac{4592}{15120}$	$\frac{45924}{151200}$
$	$\left[0,\frac{3}{11}\right)$	$\left[\frac{3}{11},\frac{6}{11}\right)$	$\left[\frac{6}{11},\frac{8}{11}\right)$	$\left[\frac{8}{11},\frac{10}{11}\right)$	$\left[\frac{10}{11},1\right)$	$\frac{4}{151200}$	$\frac{505160}{1663200}$	$\frac{45924}{151200}$

Следовательно, строку «краска$» можно представить числом $x \in \left(\frac{505160}{1663200}, \frac{505164}{1663200} \right)$, например 0,30372896. Пример окончен.

Алгоритм арифметического декодирования может быть описан следующим образом:

Исходные данные алгоритма: число, алфавит $A = \{a_1, ..., a_n\}$.

Шаг 1. Инициализация.

Введем вспомогательные переменные, аналогичные рассматриваемым в процессе компрессии.

Шаг 2. Работа алгоритма.

Пока символ не равен символу конца блока, выполняется цикл.

1. Построить таблицу распределения вероятностей

символ	вероятность	интервал	верхняя граница символа	нижняя граница символа
a_1	$\frac{n_{a_1}}{n}$	$\left[0, \frac{n_{a_1}}{n} \right)$	$\frac{n_{a_1}}{n}$	0
…	…	…	…	…
a_i	$\frac{n_{a_i}}{n}$	$\left[\frac{n_{a_{i-1}}}{n}, \frac{n_{a_{i-1}}}{n} + \frac{n_{a_i}}{n} \right)$	$\frac{n_{a_{i-1}}}{n} + \frac{n_{a_i}}{n}$	$\frac{n_{a_{i-1}}}{n}$

2. Символ = найти символ в интервал которого попадает число.

3. Выдать символ.

4. Интервал = верхняя граница (символа) – нижняя граница (символа).

5. Число = число – нижняя граница (символа);

6. Число = число / интервал.

7. $n_{b_i} = n_{b_i} + 1; \;\; n = n + 1.$

Цикл окончен. Алгоритм окончен.

Пример. Дано число 0,303729 и алфавит $A = \{"к", "р", "с", "а", "\$"\}$. Найти закодированную строку B.

Решение

Процесс декодирования можно описать следующей таблицей.

число	таблица вероятностей					интервал	нижняя граница символа	верхняя граница символа	символ
инициализация							0	1	
	«а»	«к»	«р»	«с»	«$»				
0,303729	$\left[0,\frac{1}{5}\right)$	$\left[\frac{1}{5},\frac{2}{5}\right)$	$\left[\frac{2}{5},\frac{3}{5}\right)$	$\left[\frac{3}{5},\frac{4}{5}\right)$	$\left[\frac{4}{5},1\right)$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{2}{5}$	к
0,518645	$\left[0,\frac{1}{6}\right)$	$\left[\frac{1}{6},\frac{3}{6}\right)$	$\left[\frac{3}{6},\frac{4}{6}\right)$	$\left[\frac{4}{6},\frac{5}{6}\right)$	$\left[\frac{5}{6},1\right)$	$\frac{1}{6}$	$\frac{3}{6}$	$\frac{4}{6}$	р
0,11187	$\left[0,\frac{1}{7}\right)$	$\left[\frac{1}{7},\frac{3}{7}\right)$	$\left[\frac{3}{7},\frac{5}{7}\right)$	$\left[\frac{5}{7},\frac{6}{7}\right)$	$\left[\frac{6}{7},1\right)$	$\frac{1}{7}$	0	$\frac{1}{7}$	а
0,78309	$\left[0,\frac{2}{8}\right)$	$\left[\frac{2}{8},\frac{4}{8}\right)$	$\left[\frac{4}{8},\frac{6}{8}\right)$	$\left[\frac{6}{8},\frac{7}{8}\right)$	$\left[\frac{7}{8},1\right)$	$\frac{1}{8}$	$\frac{6}{8}$	$\frac{7}{8}$	с
0,26472	$\left[0,\frac{2}{9}\right)$	$\left[\frac{2}{9},\frac{4}{9}\right)$	$\left[\frac{4}{9},\frac{6}{9}\right)$	$\left[\frac{6}{9},\frac{8}{9}\right)$	$\left[\frac{8}{9},1\right)$	$\frac{2}{9}$	$\frac{2}{9}$	$\frac{4}{9}$	к
0,19124	$\left[0,\frac{2}{10}\right)$	$\left[\frac{2}{10},\frac{5}{10}\right)$	$\left[\frac{5}{10},\frac{7}{10}\right)$	$\left[\frac{7}{10},\frac{9}{10}\right)$	$\left[\frac{9}{10},1\right)$	$\frac{2}{10}$	0	$\frac{2}{10}$	а
0,9562	$\left[0,\frac{3}{11}\right)$	$\left[\frac{3}{11},\frac{6}{11}\right)$	$\left[\frac{6}{11},\frac{8}{11}\right)$	$\left[\frac{8}{11},\frac{10}{11}\right)$	$\left[\frac{10}{11},1\right)$	$\frac{1}{11}$	$\frac{10}{11}$	1	$

В результате работы алгоритма декомпрессии получили строку «краска$». Пример окончен.

Алгоритм RLE

Данный алгоритм часто используется при сжатии изображений.

Работа алгоритма RLE (кодирование длин повторений) основана на следующей идее:

- изображение вытягивается в цепочку байтов по строкам растра;
- сжатие происходит за счет того, что в исходном изображении встречаются цепочки одинаковых байтов. Замена таких цепочек на пары (счетчик повторений, значение) уменьшает избыточность данных.

В формате PCX это реализовано следующим образом: в выходной поток пишется либо непосредственно значение пикселя (если в двух его старших разрядах не единицы):

[XX значение]

либо пара

[11 счетчик] [что повторять],

причем повторяемый символ повторяется число раз, на единицу большую счетчика.

Отметим, что возможны ситуации, когда размер сжатого файла больше размера исходного изображения.

Алгоритм кодирования длин повторений может быть очень эффективен при сжатии двоичных данных, например, черно-белых фиксированных изображений.

В этом случае работа алгоритма основана на следующей идее:

- изображение вытягивается в цепочку бит по строкам растра;
- вместо кодирования собственно данных кодированию подвергаются числа, соответствующие длинам участков, на которых данные сохраняют неизменное значение.

Пример. Дан двоичный вектор данных $X = (0111000011110000000100000001000000010000000111100011110111101111)$ длиной 64 бита.

Закодировать методом длин повторений.

Решение.

Выделим в векторе X участки, на которых данные сохраняют неизменное значение, и определим их длины. Результирующая последовательность длин участков – положительных целых чисел, соответствующих исходному вектору данных X, – будет иметь вид $r = (1,3,4,4,7,1,7,1,7,1,7,4,3,4,1,4,1,4).$ Теперь эту последовательность можно закодировать каким-либо статистическим кодом, например, кодом Хаффмана. В результате получим

длина участка	4	1	7	3
код	0	10	110	111

Поскольку кодируемая строка начинается с нуля, для однозначной декодировки добавим в начало полученного вектора символ 0. В результате получим кодовое слово $h = (0101110011010110101101011001110100100)$ длинной в 37 бит, то есть информация была сжата почти на половину. Пример окончен.

Преобразование Барроуза-Уилера

Преобразование Барроуза-Уиллера (BWT) предназначено для того, чтобы преобразовать входной блок в более удобный для сжатия вид. Впервые этот метод был опубликован в 1994 году. Заметим, что полученный после преобразования блок эффективнее сжимать специально разработанными для этого алгоритмами. Поэтому лучше рассматривать описываемый алгоритм вместе со специфическими методами кодирования данных. В оригинальной статье была предложена совокупность из трех пунктов:

1) преобразование BWT;

2) преобразование Move-To-Front (MTF перемещение стопки книг) или кодирование расстояний, или какое-нибудь аналогичное преобразование;

3) статистический кодер для сжатия данных, полученных в результате последовательного применения предыдущих преобразований.

Для понимания процесса сжатия при помощи методов, основанных на преобразовании BWT, рассмотрим каждый из этапов подробно.

Преобразование Барроуза-Уилера

Прямое преобразование

Отметим, что преобразование BWT работает сразу либо со всеми элементами входного потока, либо с достаточно большим блокам.

Процедуру преобразования можно условно разделить на четыре этапа:

1) выделить блок из входного потока;

2) сформировать матрицу всех перестановок, полученных в результате циклического сдвига блока;

3) отсортировать все перестановки в соответствии с лексикографическим порядком символов каждой перестановки;

4) на выход подается последний столбец матрицы и номер строки, соответствующий оригинальному блоку.

Пример. Рассмотрим работу преобразования BWT для строки символов «безземелье».

После выполнения второго этапа получим множество циклических перестановок.

безземелье
езземельеб
зземельебе
земельебез
емельебезз
мельебеззе
ельебеззем
льебезземе
ьебезземел
ебезземель

Пометим в этой матрице исходную строку и отсортируем все строки в лексикографическом порядке. В результате получим

0.	безземелье	исходная строка
1.	ебезземель	
2.	езземельеб	
3.	ельебеззем	
4.	емельебезз	
5.	земельебез	
6.	зземельебе	
7.	льебезземе	
8.	мельебеззе	
9.	ьебезземел	

После выполнения четвертого этапа, взяв последний столбец, получим строку «еьбмззееел», а номер исходной строки в отсортированной матрице равен нулю. Пример окончен.

Обратное преобразование

Рассмотрим идею обратного преобразования BWT на конкретном примере. Пусть после прямого преобразования у нас есть только строка «еьбмззееел», элементы которой – символы последнего столбца матрицы и номер исходной строки в отсортированной матрице, равный нулю. Достаточно отсортировать все элементы строки «еьбмззееел», чтобы получить первый столбец матрицы циклических перестановок отсортированных слева направо в соответствии с лексикографическим порядком символов ее строк.

номер строки	до преобразования	после преобразования	номер новой строки
0	б…е	еб…	1
1	е…ь	ье…	9
2	е…б	бе…	0
3	е…м	ме…	8
4	е…з	зе…	5
5	з…з	зз…	6
6	з…е	ез…	2
7	л…е	ел…	3
8	м…е	ем…	4
9	ь…л	ль…	7

Символы первого и последнего столбцов образуют пары, поскольку строки матрицы были получены в результате циклического сдвига исходной строки. Отсортировав эти пары, получим два известных столбца в левой части матрицы. Аналогично столбец за столбцом можно восстановить всю матрицу.

бе…е	без…е	безз…е	беззе…е	беззем…е
еб…ь	ебе…ь	ебез…ь	ебезз…ь	ебеззе…ь
ез…б	езз…б	еззе…б	еззем…б	езземе…б
ел…м	ель…м	елье…м	ельеб…м	ельебе…м
ем…з	еме…з	емел…з	емель…з	емелье…з
зе…з	зем…з	земе…з	земел…з	земель…з
зз…е	ззе…е	ззем…е	зземе…е	зземел…е
ль…е	лье…е	льеб…е	льебе…е	льебез…е
ме…е	мел…е	мель…е	мелье…е	мельеб…е
ье…л	ьеб…л	ьебе…л	ьебез…л	ьебезз…л
безземе…е	безземел…е	безземелье		
ебеззем…ь	ебезземе…ь	ебезземель		
езземел…б	езземель…б	езземельеб		
ельебез…м	ельебезз…м	ельебеззем		
емельеб…з	емельебе…з	емельебезз		
земелье…з	земельеб…з	земельебез		
земель…е	зземелье…е	зземельебе		
льебезз…е	льебеззе…е	льебезземе		
мельебе…е	мельебез…е	мельебеззе		
ьебеззе…л	ьебеззем…л	ьебезземел		

Учитывая количество символов в строке и полученный после прямого преобразования номер, можно однозначно определить декодированную строку.

Однако декодированную строку можно найти, не осуществляя для этого посимвольное выписывание строк матрицы перестановок, но потребуется вектор обратного преобразования T, представляющий собой массив чисел, и размер которого равен числу символов в блоке.

Для получения вектора обратного преобразования T нужно определить лишь порядок определения символов первого столбца из символов последнего. Другими словами, после того как нашли первый столбец матрицы, надо отсортировать матрицу по номерам новых строк.

номер строки		номер новой строки	
2	е…б	0	бе…е
0	б…е	1	еб…ь
6	з…е	2	ез…б
7	л…е	3	ел…м
8	м…е	4	ем…з
4	е…з	5	зе…з
5	з…з	6	зз…е
9	ь…л	7	ль…е
3	е…м	8	ме…е
1	е…ь	9	ье…л

Полученные значения номеров строк и есть искомый вектор обратного преобразования $T = \{2,0,6,7,8,4,5,9,3,1\}$. Для получения декодированной строки надо выписать символы из исходной строки «еьбмззееел» в порядке, указанном в векторе обратного преобразования T, начиная с номера строки полученного в процессе прямого преобразования BWT. Другими словами, декодирования строки можно записать:

индекс	$T[0]=2\Rightarrow$	$T[2]=6\Rightarrow$	$T[6]=5\Rightarrow$	$T[5]=4\Rightarrow$	$T[4]=8\Rightarrow$	$T[8]=3\Rightarrow$
символ	б	е	з	з	е	м

индекс	$T[3]=7\Rightarrow$	$T[7]=9\Rightarrow$	$T[9]=1\Rightarrow$	$T[1]=0$
символ	е	л	ь	е

Пример окончен.

Отметим, что главная задача преобразования Барроуза-Уилера – ловко переставить символы, чтобы их можно было легко сжать.

Преобразование Move-To-Front

Метод Move-To-Front (MTF) также известен под названием «перемещение стопки книг». Идею метода легко проиллюстрировать на процессе перемещения книг в строке: «Из стопки книг

вытаскивают нужную книгу и кладут ее сверху». Следовательно, можно ожидать, что наиболее часто используемые книги через некоторое время окажутся ближе к верху стопки.

Прямое преобразование

Рассмотрим работу метода MTF на примере строки «еьбмззееел». Легко заметить, что в строке используются только символы из алфавита $M = \{е, ь, б, м, з, л\}$. Будем считать, что начальный список MTF содержит символы алфавита M в следующем порядке $\{б, е, з, л, м, ь\}$.

символ	список MTF	выход преобразования
е	безлмь	1
ь	ебзлмь	5
б	ьебзлм	2
м	бьезлм	5
з	мбьезл	4
з	змбьел	0
е	змбьел	4
е	езмбьл	0
е	езмбьл	0
л	езмбьл	5

Таким образом, после преобразования Move-To-Front на выходе получим вектор $(1,5,2,5,4,0,4,0,0,5)$. Пример окончен.

Обратное преобразование

Обратное преобразование аналогично прямому преобразованию. Проиллюстрируем обратное преобразование на примере вектора $(1,5,2,5,4,0,4,0,0,5)$, учитывая, что начальный список MTF содержит строку «безлмь».

вход преобразования	список MTF	выход преобразования
1	безлмь	е
5	ебзлмь	ь
2	ьебзлм	б
5	бьезлм	м

4	мбьезл	з
0	змбьел	з
4	змбьел	е
0	езмбьл	е
0	езмбьл	е
5	езмбьл	л

Пример окончен.

Назначение этого метода – упростить задачу статистическому кодеру. Это связано с тем, что результат преобразования Барроуза-Уилера представляет собой последовательность символов, среди которых часто попадаются идущие подряд одинаковые символы.

Приведем иллюстрацию полезности применения Move-To-Front преобразования. Напомним, что при кодировании строки «еьбмззееел» по методу Хаффмана без использования MTF нам потребовалось 24 бита. Рассмотрим теперь кодирование этой же строки после преобразования MTF.

символ	частота	вероятность	код Хаффмана
1	1	$1/10$	001
5	3	$3/10$	11
2	1	$1/10$	000
4	2	$2/10$	01
0	3	$3/10$	10

В результате кодирования получаем последовательность длиной 1*3+3*2+1*3+2*2+3*2=22 бита.

Более наглядно полезность преобразования MTF можно рассмотреть на строке «ссссааааббддддбб». Поскольку вероятность любого символа в строке равняется $\frac{1}{4}$, то для кодирования одного символа по коду Хаффмана потребуется 2 бита или 32 бита на всю строку. Теперь рассмотрим кодирование после преобразования MTF. Предположим, что начальный список MTF «абсд», тогда

с с с с а а а а б б д д д д б б	исходная строка
2 0 0 0 1 0 0 0 2 0 3 0 0 0 1 0	выход преобразования MTF

Рассмотрим теперь кодирование преобразованной строки методом Хаффмана.

символ	частота	вероятность	код Хаффмана
0	11	$11/16$	1
2	2	$2/16$	00
1	2	$2/16$	011
3	1	$1/16$	010

В результате кодирования получаем последовательность длиной 1*11+2*2+2*3+1*3=24 бита.

Кодирование расстояний

Прямое преобразование

Этот метод является одной из альтернатив MTF. Рассмотрим процесс прямого преобразования на строке A=«еьбмззееел», заметим что алфавит «б,е,з,л,м,ь». Но перед тем как рассмотреть процесс кодирования, изменим строку A. К началу строки припишем все символы алфавита в лексикографическом порядке, а в конец строки добавим символ, обозначающий конец блока, получим A_1 =«безлмьеьбмззееел$».

строка	безлмьеьбмззееел$
известные символы	безлмь**……**.$

Теперь займемся кодированием расстояний. Берем первый из известных символов – это «б» и ищем ближайший такой же. Наша задача – закодировать номер той вакантной позиции, на которую выпадет этот символ. Расстояние (номер) находим, подсчитывая число точек, символизирующих незанятые позиции в строке известных символов.

строка	безлмьеьбмззееел$
известные символы	безлмь . . б . . . . . .$
расстояние	2

Аналогично кодируем еще несколько символов по очереди.

строка	безлмьеьбмззееел$
известные символы	безлмь е . б . . . . . .$
расстояние	20
известные символы	безлмь е . б . з . . . . .$
расстояние	20**2**
известные символы	безлмь е . б . з . . . . л$
расстояние	202**6**
известные символы	безлмь е . б м з . . . . л$
расстояние	2026**1**
известные символы	безлмь е ь б м з . . . . л$
расстояние	20261**0**
известные символы	безлмь е ь б м з **. е . .** л$
расстояние	202610**1**
известные символы	безлмьеьбмз **.** е **. .** л$
расстояние	2026101**3**

Кодируя очередной символ «ь», мы сделали ссылку на конец строки. При декодировании такая ссылка позволит понять, что символы «ь» в строке закончились. Заметим, что если рядом с рассматриваемым известным символом будет находиться вакантная позиция, то это значит, что там будет стоять такой же символ, а вместо ссылки можно поставить прочерк. Расстановка прочерков позволит уменьшить количество символов для последующего сжатия.

строка	безлмьеьбмззееел$
известные символы	безлмье ьбм з **.** е **. .** л$
расстояние	2026101333
строка	безлмьеьбмззееел$
известные символы	безлмьеьбм з**з**е **. .** л$
расстояние	2026101333-2
строка	безлмьеьбмззееел$
известные символы	безлмьеьбм з**зе е .** л$
расстояние	2026101333-2**-**
строка	безлмьеьбмззееел$
известные символы	безлмьеьбм з**зее**л$
расстояние	2026101333-2**--**

В итоге получаем последовательность $\{2,0,2,6,1,0,1,3,3,3,2\}$.

Обратное преобразование

Проиллюстрируем обратное преобразование на примере вектора {2,0,2,6,1,0,1,3,3,3,2} учитывая, что размер блока равен десяти, а алфавит имеет вид «б,е,з,л,м,ь». Следовательно, уже до момента декодирования мы знаем, что строка имеет вид «безлмь**……**$». Рассмотрим более подробно процесс декодирования

известные символы	«безлмь**. .б…….**$»
расстояние	**2**,0,2,6,1,0,1,3,3,3,2
известные символы	«безлмь**е .**б**…….**$»
расстояние	2,**0**,2,6,1,0,1,3,3,3,2
известные символы	«безлмье **.**б**. з . ….**$»
расстояние	2,0,**2**,6,1,0,1,3,3,3,2
известные символы	«безлмье **.**б**.** з **. ….л**$»
расстояние	2,0,2,**6**,1,0,1,3,3,3,2
известные символы	«безлмье **.**б**м** з **. …**л$»
расстояние	2,0,2,6,**1**,0,1,3,3,3,2
известные символы	«безлмье **ь**бм з **. …**л$»
расстояние	2,0,2,6,1,**0**,1,3,3,3,2
известные символы	«безлмье ьбм з **. е . .** л$»
расстояние	2,0,2,6,1,0,**1**,3,3,3,2
известные символы	«безлмье ьбм з **.** е **. .** л$»
расстояние	2,0,2,6,1,0,1,**3**,3,3,2
известные символы	«безлмье ьбм з **.** е **. .** л$»
расстояние	2,0,2,6,1,0,1,3,**3**,3,2
известные символы	«безлмье ьбм з **.** е **. .** л$»
расстояние	2,0,2,6,1,0,1,3,3,**3**,2
известные символы	«безлмье ьбм з **з** е **. .** л$»
расстояние	2,0,2,6,1,0,1,3,3,3,2
известные символы	«безлмье ьбм з з е **е е** л$»
расстояние	2,0,2,6,1,0,1,3,3,3,2

Более наглядно полезность преобразования можно проиллюстрировать на строке «ссссааааббдддддбб». Поскольку алфавит, используемый в строке «абсд», то легко заметить, что после кодирования расстояний будет получен вектор (4,7,0,7,9,6,3,1)

Рассмотрим теперь кодирование преобразованной строки методом Хаффмана.

символ	код Хаффмана	символ	код Хаффмана
7	10	4	011
0	000	6	110
1	001	9	111
3	010		

Если исходная строка была длиной 2*16=32 бита, то в результате преобразования и кодирования получим последовательность длиной 2*2+3*6=22 бита.

Методы сжатия с потерей информации

Часто нет необходимости в абсолютной точности передачи исходных данных получателю. Это может быть связано с тем, что получатели информации – органы чувств человека, исполнительные механизмы и т. д. – обладают конечной разрешающей способностью, то есть не замечают незначительной разницы между *абсолютно точным* и *приближенным* значениями воспроизводимого сообщения. Порог чувствительности к искажениям также может быть различным, но он всегда есть.

Большинство методов сжатия с потерями основано на кодировании не самих данных, а некоторых линейных преобразований от них.

Отметим, что в смысле упаковки методы сжатия с потерями более эффективны, чем методы сжатия без потерь. Хорошим примером сжатия информации с потерями является стандарт сжатия изображения jpeg.

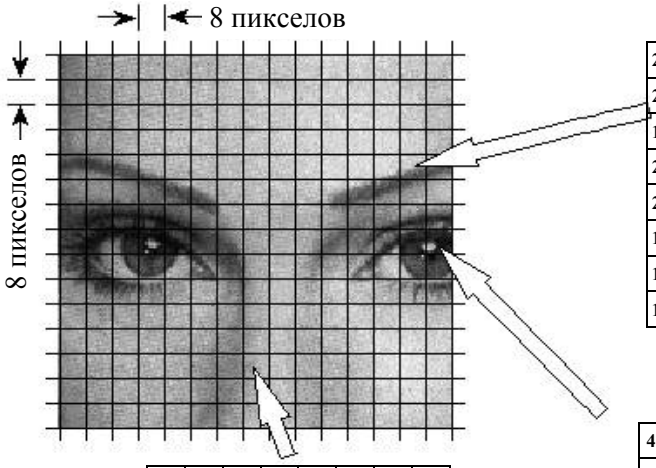
Стандарт сжатия JPEG

Принцип сжатия изображений. Если случайно выбрать пиксель изображения, то с большой вероятностью ближайшие к нему пикселы будут иметь тот же или близкий цвет.

Опишем основные шаги алгоритма ***JPEG.***

Рассмотрим работу алгоритма сжатия ***JPEG*** при кодировании полутонового черно-белого изображения с числом градаций яркости в 256 уровней (8 двоичных разрядов).

Шаг 1. Разобьем изображения на блоки размером 8×8 пикселов. Если число строк или столбцов не кратно 8, то самая нижняя строка и самый правый столбец повторяются нужное число раз. Выбор размера блока обусловлен тем фактом, что при большем размере свойства пикселов внутри блока будут сильно различаться, а при меньшем размере эффект от кодирования уменьшится.



231	224	224	217	217	203	189	196
210	217	203	189	203	224	217	224
196	217	210	224	203	203	196	189
210	203	196	203	182	203	182	189
203	224	203	217	196	175	154	140
182	189	168	161	154	126	119	112
175	154	126	105	140	105	119	84
154	98	105	98	105	63	112	84

154	154	175	182	189	168	217	175
154	147	168	154	168	168	196	175
175	154	203	175	189	182	196	182
175	168	168	168	140	175	168	203
168	168	154	196	175	189	203	154
168	161	161	168	154	154	189	189
147	161	175	182	189	175	217	175
175	175	203	175	189	175	175	182

42	28	35	28	42	49	35	42
49	49	35	28	35	35	35	42
42	21	21	28	42	35	42	28
21	35	35	42	42	28	28	14
56	70	77	84	91	28	28	21
70	126	133	147	161	91	35	14
126	203	189	182	175	175	35	21
49	189	245	210	182	84	21	35

Шаг 2. Применим к каждому из блоков дискретное косинусное преобразование (DCT).

Дискретное косинусное преобразование от изображения P, вычисляемое кодером, может быть записано следующим образом:

$$G_{ij} = \frac{1}{4} C_i C_j \sum_{x=0}^{7} \sum_{y=0}^{7} p_{xy} \cos\left(\frac{(2y+1)j\pi}{16}\right) \cos\left(\frac{(2x+1)i\pi}{16}\right),$$

где $C_k = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & k > 0 \end{cases}$

В результате получим двумерный спектр G, также имеющий размер 8×8.

Декодер JPEG вычисляет обратное преобразование DCT

$$p_{xy} = \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} C_i C_j G_{ij} \cos\left(\frac{(2y+1)j\pi}{16}\right) \cos\left(\frac{(2x+1)i\pi}{16}\right),$$

где $C_k = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & k > 0 \end{cases}$

Заметим, что преобразование DCT удобнее представлять в виде произведения матриц. Прямое преобразование будет выглядеть как $G = MPM^T$, обратное – $P = M^T GM$,

где $M_{ij} = \begin{cases} \frac{1}{\sqrt{8}}, & i = 0 \\ \frac{1}{2}\cos\left(\frac{(2j+1)i\pi}{16}\right), & i > 0 \end{cases}$

Пример. Пусть блок изображения имеет следующие значения.

42	28	35	28	42	49	35	42
49	49	35	28	35	35	35	42
42	21	21	28	42	35	42	28
21	35	35	42	42	28	28	14
56	70	77	84	91	28	28	21
70	126	133	147	161	91	35	14
126	203	189	182	175	175	35	21
49	189	245	210	182	84	21	35

После выполнения прямого преобразования DCT получим таблицу.

-466	133	-160	-21	-11	-51	-6	2
-299	-140	174	54	26	52	4	12
126	40	-41	-33	-25	-27	22	-12
33	8	-8	34	10	-5	2	-3
-54	-11	-4	-48	7	6	-6	17
20	-4	1	44	-18	0	18	-24
-20	-19	-22	-17	14	7	-9	20
14	5	15	18	5	5	8	-2

(перед перемножением матриц из значений яркости вычитаем число 128, благодаря чему значения смещаются в диапазон -128…+127). Легко заметить, что полученная матрица обладает следующим свойством: коэффициенты с большими абсолютными значениями концентрируются в верхнем левом углу. Можно сказать, что в левом верхнем углу размещаются самые важные данные, а в правом нижнем – наименее важные.

-466	133	-160	-21	-11	-51	-6	2
-299	-140	174	54	26	52	4	12
126	40	-41	-33	-25	-27	22	-12
33	8	-8	34	10	-5	2	-3
-54	-11	-4	-48	7	6	-6	17
20	-4	1	44	-18	0	18	-24
-20	-19	-22	-17	14	7	-9	20
14	5	15	18	5	5	8	-2

После выполнения обратного преобразования DCT и прибавления числа 128 получим таблицу.

42	28	35	28	41	49	35	42
49	49	35	27	35	35	35	41
42	21	21	29	42	35	42	28
21	35	35	42	42	29	28	14
56	70	77	84	91	28	28	20
70	126	133	147	161	91	35	14
126	203	189	182	175	175	35	21
49	189	245	210	181	84	21	35

Легко заметить, что на этом шаге потери незначительные и происходят только от ограниченности точности машинных вычислений. Пример окончен.

Шаг 3. Квантование.

Каждое число из матрицы коэффициентов DCT делим на специальное число из «таблицы квантования», а результат округляем до ближайшего целого. Легко заметить, что чем больше числа, на которые происходит деление, тем больше в результате деления будет нулевых значений, а значит, сильнее сжатие и за-

метнее потери. Значения «таблицы квантования» являются изменяемыми параметрами, которые, в принципе, можно менять. Например, вычисляется простая «таблица квантования» с элементами вида $Q_{ij} = 3 + (i + j) * R,$ легко заметить, что они зависят от параметра $R,$ который задается пользователем.

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

таблица квантования при $R = 2$

Однако можно воспользоваться и «готовыми» таблицы квантования.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

таблица для светимости

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

таблица для цветности

Если квантование сделано правильно, то в полученной на этом шаге таблице останется всего несколько ненулевых коэффициентов, которые будут сконцентрированы в левом верхнем углу матрицы.

Шаг 4. Преобразуем матрицу $8 \times 8,$ полученную на предыдущем шаге, в линейную последовательность при помощи зигзаг-сканирования, т. е. берем элементы с индексами (0,0), (0,1), (1,0) (2,0), (1,1), (0,2) и так далее.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)

(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

Шаг 5. Кодирование длин повторений для сокращения числа нулевых компонент.

Шаг 6. Кодирование результата шага 5 кодом Хаффмана.

Шаг 7. На последнем шаге добавляется заголовок из использованных параметров ***JPEG*** и результат выводится в сжатый файл. Алгоритм закончен.

Перед тем как мы завершим обсуждение алгоритма, необходимо рассмотреть, как изменится его работа при сжатии цветного изображения. Предположим, что сжимаем 24 битовое изображение, тогда в рассмотренном алгоритме добавится два шага:

Шаг 0. Цветное изображение из пространства *RGB* преобразуется в цветовое пространство *YCrCb* (где *Y* – яркостная составляющая, а *Cr, Cb* – компоненты, отвечающие за цвет хроматический красный и хроматический синий). Цветовое пространство *YCrCb* иногда называют *YUV*. Переход к *YCrCb* обусловлен тем фактом, что глаз чувствителен к малым изменениям яркости пикселов, но не цветности, поэтому из компоненты цветности можно удалить значительную долю информации для достижения высокого сжатия без заметного визуального ухудшения качества образа. Преобразование можно задать с помощью следующей формулы:

$$\begin{pmatrix} Y \\ C_r \\ C_b \end{pmatrix} = \begin{pmatrix} 0{,}2990 & 0{,}5870 & 0{,}1140 \\ 0{,}5000 & -0{,}4187 & -0{,}0813 \\ -0{,}1687 & -0{,}3313 & 0{,}5000 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix},$$

а обратное преобразование

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1{,}402 \\ 1 & -0{,}34414 & -0{,}71414 \\ 1 & 1{,}772 & 0 \end{pmatrix} * \left(\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} - \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} \right).$$

Затем цветное изображение может быть разбито на крупные пикселы (опускается для полутоновых изображений и не делается для компоненты яркости). Укрупнение пикселов делается или в соотношении 2:1 по горизонтали и вертикали (так называемое укрупнение 2h2v) или в пропорциях 2:1 по горизонтали и 1:1 по вертикали (укрупнение 2hlv).

Дальнейшие шаги алгоритма аналогичны случаю, рассмотренному выше.

Вейвлетные методы

Вычисление средних и полуразностей

Рассмотрим вначале одномерный массив данных, состоящий из N элементов. Причем будем считать, что число $N = 2^k$. Идея алгоритма заключается в том, что будем сохранять в файл разницу – число между средними значениями соседних блоков в изображении, которая обычно близка к нулю. Так, два числа a_{2i} и a_{2i+1} всегда можно представить в виде чисел $b_i^1 = (a_{2i} + a_{2i+1})/2$ и $b_i^2 = (a_{2i} - a_{2i+1})/2.$

Пример. Рассмотрим массив чисел $a_i = (1,3,4,6,8,10,14,18).$ Сначала вычислим четыре средние величины $b_i^1 = (2,5,9,16)$ и четыре полуразности $b_i^2 = (-1,-1,-1,-2).$ В результате получим вектор $(2,5,9,16,-1,-1,-1,-2).$ Повторим рассматриваемую процедуру применительно к четырем первым (крупным) компонентам нового массива. Они преобразуются в два средних и в две полуразности. Остальные четыре компонента оставим без изменений. В результате получим массив $(3.5,12.5,-1.5,-3.5,-1,-1,-1,-2).$ При следующей итерации получим массив чисел $(8,-4.5,-1.5,-3.5,-1,-1,-1,-2)$, который называется вейвлетным преобразованием Хаара исходного массива данных. Заметим, что по полученному вектору легко восстанавливается первоначальный массив.

Операции, которые лежат в основе преобразования Хаара, можно выразить с помощью соответствующих матриц. Так, первую итерацию можно описать с помощью матрицы A_1. Аналогично матрицы A_2 и A_3 производят соответственно второй и третий шаг преобразования.

$$A_1=\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{-1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{-1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{-1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{-1}{2} \end{pmatrix} A_2=\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{-1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{-1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} A_3=\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{-1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Следовательно, преобразование можно задать формулой вида $I_{tr}=WI,$ где I – исходная строка, а $W=A_3A_2A_1$. Обратное преобразование тогда будет $I=W^{-1}I_{tr}$. Пример окончен

Полученный вектор можно использовать для сжатия с помощью какого-либо из ранее рассмотренных способов.

Перенесем предложенное преобразование на двумерный случай. Преобразование будем осуществлять для блока размером $N\times N,$ где $N=2^k$. Существует много обобщений этого преобразования, но мы рассмотрим только пирамидальное разложение. Будем вычислять рассмотренное преобразование, применяя итерации поочередно к строкам и столбцам. На первом шаге вычисляются полусуммы и полуразности для всех строк (только одна итерация, а не всё преобразование). Это действие производит средние в левой половине матрицы и полуразности – в правой половине. На втором шаге вычисляются полусуммы и полуразности для всех столбцов получившейся матрицы.

Пример. Пусть задано начальное изображение в виде матрицы размера 8×8.

12	8	12	16	12	20	10	14
12	16	14	18	12	8	8	12
12	16	12	16	12	16	12	16
8	12	12	16	16	20	20	24

8	16	12	16	12	20	10	14
10	10	10	10	10	10	10	10
18	10	14	14	18	14	12	16
14	14	14	14	16	16	16	16

Тогда результаты двух первых шагов пирамидального преобразования будут выглядеть следующим образом.

10	14	16	12	2	-2	-4	-2
14	16	10	10	-2	-2	2	-2
14	14	14	14	-2	-2	-2	-2
10	14	18	22	-2	-2	-2	-2
12	14	16	12	-4	-2	-4	-2
10	10	10	10	0	0	0	0
14	14	16	14	4	0	2	-2
14	14	16	16	0	0	0	0
первый шаг для строк							

12	15	13	11	0	-2	-1	-2
12	14	16	18	-2	-2	-2	-2
11	12	13	11	-2	-1	-2	-1
14	14	16	15	2	0	1	-1
-2	-1	3	1	2	0	-3	0
2	0	-2	-4	0	0	0	0
1	2	3	1	-2	-1	-2	-1
0	0	0	-1	2	0	1	-1
второй шаг для столбцов							

В результате получаем матрицу, состоящую из четырех подматриц. В первой хранится уменьшенная копия исходного изображения, а три остальных показывают детали изображения. Аналогичное пирамидальное преобразование можно выполнить с первой подматрицей, взяв ее в качестве исходного изображения.

Заметим, что если на основе данного преобразования будем строить алгоритм сжатия с потерями, то коэффициенты четвертой подматрицы можно квантовать достаточно грубо без существенных потерь качества, а коэффициенты первой подматрицы следует квантовать очень слабо. Пример окончен.

Алгоритм JPEG 2000

Алгоритм был создан в 2000 году. Алгоритм может быть использован как для сжатия с потерями, так и для сжатия без потерь. Основная схема алгоритма JPEG 2000 очень похожа на схему алгоритма JPEG. Однако есть некоторые отличия, которые позволяют достигать большего сжатия с меньшими потерями в качестве изображения. Отличия заключаются в следующем:

• дискретное косинусное преобразование было заменено на дискретное wavelet -преобразование;

• вместо кодирования по Хаффману используется арифметическое кодирование.

Опишем основные шаги алгоритма ***JPEG2000.***

Шаг 1. Сдвиг по яркости.

Сдвиг по яркости осуществляется для каждой компоненты (RGB) изображения перед преобразованием в YUV. Это делается для выравнивания динамического диапазона, что приводит к увеличению степени сжатия.

$$I'(x, y) = I(x, y) - 2^{s-1}.$$

Значение степени s для каждой компоненты свое. При восстановлении изображения выполняется обратное преобразование:

$$I(x, y) = I'(x, y) + 2^{s-1}.$$

Шаг 2. Переход в другое цветовое пространство.

Изображение переводится из цветового пространства RGB в цветовое пространство YUV. Этот шаг аналогичен тому, какой проводится в алгоритме JPEG.

Отличие появляется, если рассматривать сжатее без потери информации, тогда переход в пространство YUV будет осуществляться по формуле

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} \left\lfloor \dfrac{R+2G+B}{4} \right\rfloor \\ R-G \\ B-G \end{pmatrix},$$

а обратное преобразование осуществляется с помощью

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} U+G \\ Y-\left\lfloor \dfrac{U+V}{4} \right\rfloor \\ V+G \end{pmatrix}$$

Шаг 3. Дискретное wavelet-преобразование (DWT).

Коэффициенты преобразования (для сжатия с потерями) задаются следующим образом:

j	Коэффициенты при упаковке		Коэффициенты при распаковке	
	Низкочастотные коэффициенты $h_L(j)$	Высокочастотные коэффициенты $h_H(j)$	Низкочастотные коэффициенты $g_L(j)$	Высокочастотные коэффициенты $g_H(j)$
0	1,115087052456994	0,6029490182363579	0,6029490182363579	1,115087052456994
± 1	0,5912717631142470	-0,2668641184428723	-0,2668641184428723	0,5912717631142470

± 2	-0,05754352622849957	-0,07822326652898785	-0,07822326652898785	-0,05754352622849957
± 3	-0,09127176311424948	0,01686411844287495	0,01686411844287495	-0,09127176311424948
± 4	0	0,02674875741080976	0,02674875741080976	0
другие j	0	0	0	0

Коэффициенты преобразования (для сжатия без потерь):

j	Коэффициенты при упаковке		Коэффициенты при распаковке	
	Низкочастотные коэффициенты $h_L(j)$	Высокочастотные коэффициенты $h_H(j)$	Низкочастотные коэффициенты $g_L(j)$	Высокочастотные коэффициенты $g_H(j)$
0	$6/8$	1	1	$6/8$
± 1	$2/8$	$-1/2$	$1/2$	$-2/8$
± 2	$-1/8$	0	0	$-1/8$

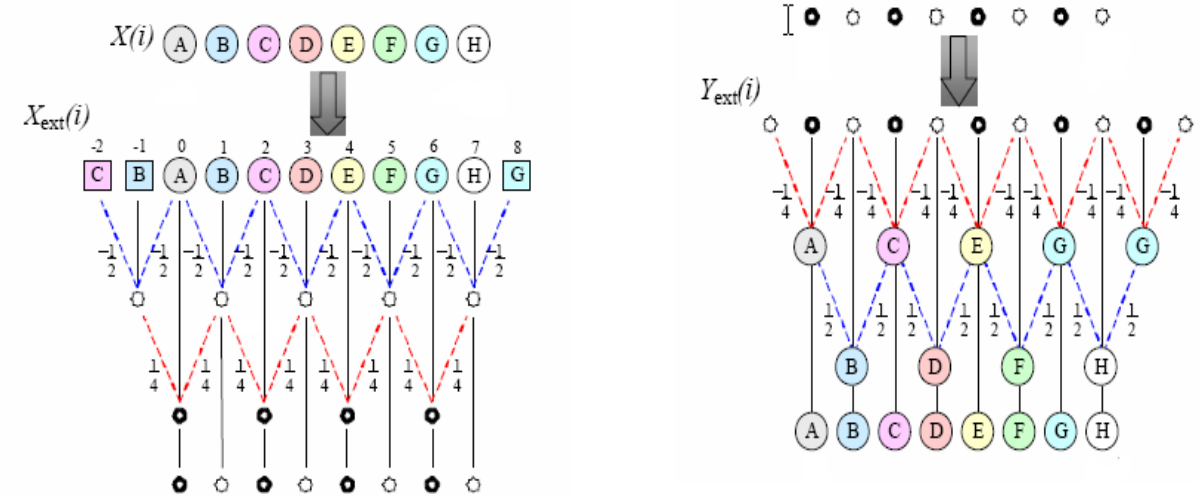
Само преобразование в одномерном случае представляет собой скалярное произведение коэффициентов фильтра на строку преобразуемых значений x. При этом четные выходящие значения формируются с помощью низкочастотного преобразования, а нечетные – с помощью высокочастотного:

$$y(2n)=\sum_{j=0}^{N-1} x(j)*h_L(j-2n),$$

$$y(2n+1)=\sum_{j=0}^{N-1} x(j)*h_H(j-2n-1).$$

Для того чтобы преобразование можно было применять к крайним пикселам изображения, оно симметрично достраивается в обе стороны на несколько пикселов.

Пример. Рассмотрим, как работает DWT преобразование для сжатия без потерь. Пусть $x=(8,2,4,1,6,9,11,3)$, следующие шаги преобразования лучше выполнять по следующим схемам



DWT при упаковке DWT при распаковке

Используя схему упаковки, получим

X			8	2	4	1	6	9	11	3	
X_{ext}	4	2	8	2	4	1	6	9	11	3	11
шаг 1		-4		-4		-4		1		-8	
шаг 2			6		2		5		9		
Y			6	-4	2	-4	5	1	9	-8	

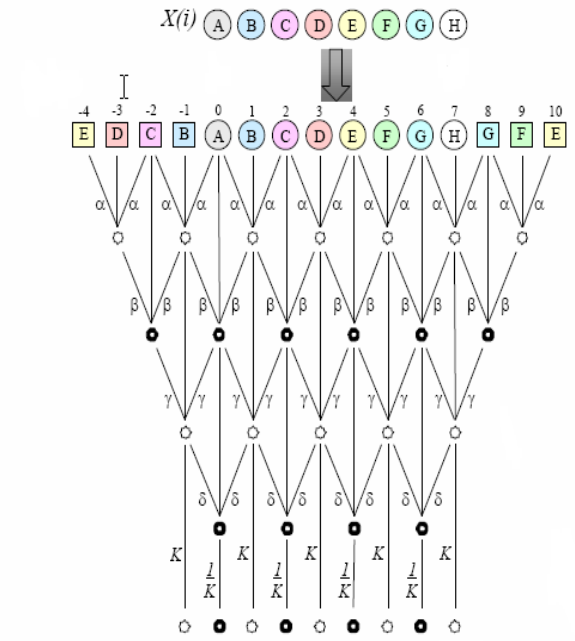
Обратное преобразование можно проиллюстрировать с помощью следующей таблицы

Y		6	-4	2	-4	5	1	9	-8		
Y_{ext}	-4	6	-4	2	-4	5	1	9	-8	9	1
шаг 1		8		4		6		11		11	
шаг 2			2		1		9		3		
X		8	2	4	1	6	9	11	3		

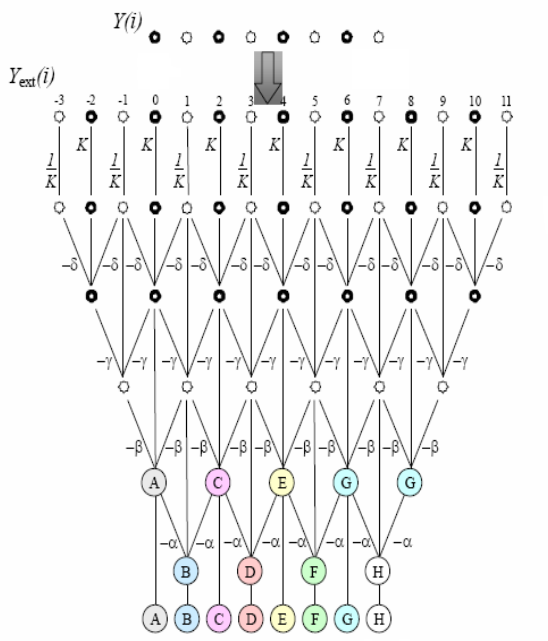
Пример окончен.

Пример. Рассмотрим, как работает DWT преобразование для сжатия с потерями.

Пусть $x = (8,2,4,1,6,9,11,3)$, тогда аналогично случаю сжатия без потерь преобразование DWT будем выполнять с помощью следующих схем:



DWT при упаковке



DWT при распаковке

где $\alpha = -1.586134342059924$, $\beta = -0.052980118572961$, $\gamma = 0.882911075530934$, $\delta = 0.443506852043971$, $K = 1.230174104914001$

Рассмотрим преобразование DWT с помощью следующей таблицы

номер элемента	-4	-3	-2	-1	0	1	2
X					8	2	4
X_{ext}	6	1	4	2	8	2	4
шаг 1		-14,861		-17,034		-17,034	
шаг 2			5,6898		9,80489		5,6898
шаг 3				-3,3532		-3,3532	
шаг 4					6,83057		2,86998
шаг 5					5,55252	-4,125	2,33299
Y					5,55252	-4,125	2,33299

номер эле-мента	3	4	5	6	7	8	9	10
X	1	6	9	11	3			
X_{ext}	1	6	9	11	3	11	9	6

шаг 1	-14,861		-17,964		-31,895		-17,964	
шаг 2		7,73911		13,6415		13,6415		
шаг 3	-3,0048		0,91293		-7,8064			
шаг 4		6,81134		10,5842				
шаг 5	-3,6964	5,53689	1,12307	8,60386	-9,6032			
Y	-3,6964	5,53689	1,12307	8,60386	-9,6032			

Обратное преобразование тогда будет

номер элемента	-3	-2	-1	0	1	2	3
Y				5,55252	-4,125	2,33299	-3,6964
Y_{ext}	-3,6964	2,33299	-4,125	5,55252	-4,125	2,33299	-3,6964
шаг 1	-3,0048	2,86998	-3,3532	6,83057	-3,3532	2,86998	-3,0048
шаг 2		5,6898		9,80489		5,6898	
шаг 3			-17,034		-17,034		14,861
шаг 4				8		4	
шаг 5					2		1
X				8	2	4	1

номер эле-мента	4	5	6	7	8	9	10	11
Y	5,53689	1,12307	8,60386	-9,6032				
Y_{ext}	5,53689	1,12307	8,60386	-9,6032	8,60386	1,12307	5,53689	-3,6964
шаг 1	6,81134	0,91293	10,5842	-7,8064	10,5842	0,91293	6,81134	-3,0048
шаг 2	7,7391		13,6415		13,6415		7,7391	
шаг 3		-17,964		-31,895		-17,964		
шаг 4	6		11		11			
шаг 5		9		3				
X	6	9	11	3				

Пример окончен.

Далее к строке применяется чересстрочное преобразование, то есть все четные коэффициенты переписываются в начало строки, а все нечетные – в конец. Это преобразование применяется сначала ко всем строкам изображения, а затем ко всем столбцам изображения. В результате изображение делится на 4 квадранта (аналогичные тем, которые были рассмотрены в методе преобразования Хаара).

Шаг 4. Квантование.

Коэффициенты, полученные после преобразования DWT, делятся на заранее заданное число (чем больше это число, тем больше коэффициентов, близких к нулю, и значит больше степень сжатия и больше потери).

Шаг 5. Сжатие полученных данных каким-нибудь статистическим алгоритмом, например арифметическим алгоритмом.

SPIHT

Метод SPIHT был разработан для оптимальной прогрессирующей передачи изображений, а также для их сжатия. Отметим, что на любом этапе декодирования качество отображаемой в этот момент картинки является наилучшим для введенного объема информации о данном образе.

Опишем основные шаги кодера SPIHT:

Шаг 1. Инициализация.

Для заданного изображения надо вычислить его вейвлетное преобразование (выбранное пользователем), разложить его на коэффициенты преобразования c_{ij} и представить их в виде целых чисел фиксированной разрядности. Предположим, что коэффициенты представлены в виде целых чисел со знаком фиксированной разрядности, например m_0, причем самый левый бит является знаковым, а в остальных $m_0 - 1$ двоичных разрядах записан модуль этого числа. Следовательно, числа меняются от $-(2^{m_0} - 1)$ до $(2^{m_0} - 1)$. Присвоим переменной n значение $\lfloor \log_2 \max_{i,j}(|c_{ij}|) \rfloor$.

Шаг 2. Сортировка.

Передать число l коэффициентов c_{ij}, которые удовлетворяют неравенству $2^n \le |c_{ij}| < 2^{n+1}$. Затем передать l пар координат и l знаков этих коэффициентов.

Шаг 3. Поправка.

Передать $(n+1)-ые$ самые старшие биты всех коэффициентов, удовлетворяющих неравенству $|c_{ij}| > 2^n$.

Шаг 4. Итерация.

Уменьшить n на 1, если необходимо сделать еще одну итерацию и перейти на Шаг 2.

Обычно последняя итерация совершается при $n = 0$, но кодер может остановиться и раньше. В этом случае наименее важная часть информации (некоторые менее значимые биты всех вейвлетных коэффициентов) не будет передаваться. В этом заключается естественное отбрасывание части информации в методе SPIHT.

Пример. Предположим, что вейвлетное преобразование изображения уже выполнено и полученные коэффициенты в виде целых чисел разрядности 16 отсортированы по абсолютной величине и хранятся в массиве m, причем элемент $m(k)$ содержит координаты (i, j) коэффициента c_{ij} так, что $\left|c_{m(k)}\right| \geq \left|c_{m(k+1)}\right|$ при всех k.

Двоичное представление упорядоченных коэффициентов

k		1	2	3	4	5	6	7	8
	знак	s	s	s	s	s	s	s	s
ст. бит	14	1	0	0	0	0	0	0	
	13	a	1	1	1	0	0	0	
	12	b	e	h	v	1	1	0	
	11	c	f	i	u	r	w	1	
	…								
мл. бит	0	d	g	j	q	t	x	p	
$m(k)$	2,3	3,4	3,2	4,4	1,2	1,1	1,3		

Будем считать, что первый коэффициент $c_{m(1)} = c_{23} = s1ab\ldots d$ (где $s, a, \ldots$ это битты)

Работу метода можно описывать с помощью следующей таблицы

номер итера-	n	l	передача пары координат	биты поправки	какой предполагаемый

ции			и знаки коэф-фициентов		результат
1	14	1	(2,3),s		$c_{23} = s1000\ldots0$
2	13	3	(3,4), (3,2), (4,4), s,s,s	a	$c_{23} = s1a00\ldots0$ $c_{34} = s0100\ldots0$ $c_{32} = s0100\ldots0$ $c_{44} = s0100\ldots0$
3	12	2	(1,1), (1,3), s,s	b,e,h,v	$c_{23} = s1ab0\ldots0$ $c_{34} = s01e0\ldots0$ $c_{32} = s01h0\ldots0$ $c_{44} = s01v0\ldots0$ $c_{11} = s0010\ldots0$ $c_{13} = s0010\ldots0$

Аналогично выполняются остальные проходы цикла до достижении нужной точности изображения или до передачи всех бит всех коэффициентов.

Пример окончен.

Задачи

1. Закодировать, используя статистический и динамический метод Хаффмана, слова «дерево», «железо», «кирпич».

2. Используя динамическое арифметическое кодирование, закодировать слова «дерево», «железо», «кирпич».

3. Используя BWT и метод «стопка книг», сделать прямое и обратное преобразования для слов «дерево», «железо», «воевода».

4. Используя BWT и метод расстояний, сделать прямое и обратное преобразования слов «дискуссия», «ассамблея», «воевода».

5. Задан вектор $x = \{1,0,0,0,0,0,0,1,1,1,1,1,1,0,1\}$, выполнить для него преобразование RLE, посмотреть, есть ли выгода от RLE, если результат преобразования или сам вектор x сжимается статистическим методом Хаффмана.

6. Предположим, что нам известен блок данных размера 8×8.

211	224	224	197	217	203	189	190
212	217	200	189	200	226	217	224
196	217	210	224	203	203	196	189
200	203	196	200	182	200	182	200
203	226	203	217	196	175	154	140
182	189	168	189	154	170	119	118
175	154	126	105	140	105	119	90
154	98	105	100	105	100	112	84

Выполнить для него преобразование Хаара.

Предположим, что нам известен блок размера 8×8 полутонового черно-белого изображения.

231	224	224	217	217	203	189	196
210	217	203	189	203	224	217	224
196	217	210	224	203	203	196	189
210	203	196	203	182	203	182	189
203	224	203	217	196	175	154	140
182	189	168	161	154	126	119	112
175	154	126	105	140	105	119	84
154	98	105	98	105	63	112	84

Выполнить для него преобразование JPEG.

8. Предположим, что нам известен вектор $x = \{10,18,20,21,44,24,11,24\}$. Выполнить для него преобразование DWT для сжатия с потерями и без потерь.

Рекомендуемая литература

1. Ватолин, Д. Методы сжатия данных / Д. Ватолин, А. Ратушняк. – М.: Диалог-Мифи, 2002.

2. Лидовский, В. В. Теория информации: учеб. пособие / В. В. Лидовский. – М.: Компания Спутник+, 2004.

3. Шульгин, В. И. Основы теории передачи информации. Ч. I. Экономное кодирование: учеб. пособие / В. И. Шульгин. – Харьков, 2003.

4. Сэломон, Д. Сжатие данных, изображений и звука / Д. Сэломон. – М.: Техносфера, 2004.

5. Гонсалес, Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. – М.: Техносфера, 2005.

6. Уэлстид, С. Фракталы и вейвлеты для сжатия изображений в действии / С. Уэлстид. – М.: Триумф, 2003.

7. Bandwidth compression (BWC) guide for JPEG 2000 visually lossless and numerically lossless compression of imagery data 2004.

Оглавление

Введение **3**

Методы сжатия без потерь **4**

Алгоритм Хаффмана *5*

Динамический алгоритм Хаффмана *8*

Адаптированное арифметическое сжатие *10*

Алгоритм RLE *14*

Преобразование Барроуза-Уилера **15**

Преобразование Барроуза-Уилера *16*

Преобразование Move-To-Front *19*

Методы сжатия с потерей информации **25**

Стандарт сжатия JPEG *25*

Вейвлетные методы **30**

Вычисление средних и полуразностей *30*

Алгоритм JPEG 2000 ..... *32*

SPIHT *38*

Задачи **41**

Рекомендуемая литература **42**

Учебное издание

Методы сжатия

Методические указания

Составитель
Краснов Михаил Владимирович

Редактор, корректор И. В. Бунакова
Компьютерная верстка Е. Л. Шелеховой

Подписано в печать 16.07.09. Формат 60×84 $^{1}/_{16}$.
Бум. офсетная. Гарнитура "Times New Roman".
Усл. печ. л. 2,56. Уч.-изд. л. 1,51.
Тираж 100 экз. Заказ .

Оригинал-макет подготовлен
в редакционно-издательском отделе Ярославского
государственного университета им. П. Г. Демидова.

Отпечатано на ризографе.

Ярославский государственный университет
им. П. Г. Демидова.
150000, Ярославль, ул. Советская, 14.

Методы сжатия