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

Методы сжатия информации: текст и изображение

Методические указания

Рекомендовано
Научно-методическим советом университета
для студентов, обучающихся по направлению
Фундаментальная информатика и информационные технологии

Ярославль
ЯрГУ
2014

УДК 681.3(072)
ББК В182я73
М54

Рекомендовано
Редакционно-издательским советом университета
в качестве учебного издания. План 2014 года

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

Составитель
М. В. Краснов

М54 **Методы сжатия информации : текст и изображение :** метод. указания / сост. М. В. Краснов ; Яросл. гос. ун-т им. П. Г. Демидова. – Ярославль : ЯрГУ, 2014. – 56 с.

Основное использование вычислительной техники связано с хранением и передачей информации, вследствие чего возникает задача об экономном методе ее записи. Описанию некоторых математических понятий и приемов, используемых при решении этой задач, и посвящена данная работа.

Методические указания предназначены для студентов, обучающихся по направлению 010300.62 Фундаментальная информатика и информационные технологии (дисциплина «Методы сжатия», цикл Б3), очной формы обучения.

УДК 681.3(072)
ББК В182я73

1. Введение

В настоящее время электронная вычислительная техника применяется во многих сферах человеческой деятельности. Основное использование вычислительной техники связано с хранением информации и организацией доступа к ней. К сожалению, основная часть данных, с которыми работают пользователи, часто представлена не компактно (обычно данные хранятся в форме, которая обеспечивает наиболее простое их использование, например текстовый документ хранится в кодировке одной из кодовых страниц Unicode, ASCII и так далее; каждый символ кодируется кодом фиксированной длины, в ASCII это 8 бит). Цель сжатия данных – обеспечить компактное представление данных, вырабатываемых источником, для более экономного их хранения и передачи по каналам связи.

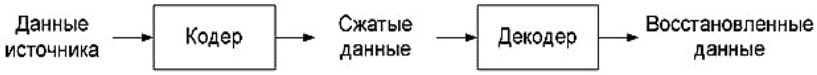


Рис. 1. Процесс работы со сжатыми данными

В этой схеме данные, вырабатываемые источником, определим как данные источника, а их компактное представление – как сжатые данные. Отметим, что под информацией понимаем значение или изменение некоторой физической величины, отражающее состояние объекта (системы или явления). Первичными сообщениями, которые генерируются источником, являются: текст, речь, музыка, изображения и т. д.

Студенты должны обратить внимание, что существуют разные модели источников данных, например источник данных без памяти и источник данных с памятью. При использовании модели источника данных без памяти события считаются независимыми, а вероятность наступления каждого события считается фиксированной величиной, т. е. $P(s_i \to s_j) = P(s_j)$. Однако события могут оказаться коррелированными друг с другом, т. е. $P(s_i \to s_j) \neq P(s_j)$, и значит надо рассматривать источник данных с памятью. Можно говорить, что источник без памяти порождает элементы, а источник

данных с памятью – «слова», поскольку во втором случае учет значений соседних элементов (контекста) улучшает сжатие.

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. Другими словами, после выполнения операции декодирования получается поток, в точности совпадающий с потоком данных источника. Обратимое сжатие обычно используется при кодировании текстовой информации.

Под необратимым сжатием, или сжатием с потерями, подразумевают такое преобразование входного потока данных, в результате которого декодер не в состоянии восстановить данные источника в первоначальном виде. Студенты должны обратить особое внимание на тот факт, что полученные при декодировании данные с некоторой точки зрения могут быть похожи по внешним характеристикам на данные источника. Обычно сжатие с потерями состоит из двух этапов:

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

b) сжатие без потерь.

Необратимое сжатие обычно используется при кодировании графической, звуковой и видеоинформации.

Студентам следует обратить внимание, что сжатию могут подвергаться не только сами исходные данные, но и какие-либо преобразования над ними.

При решении задачи сжатия естественным является вопрос, насколько эффективна та или иная система сжатия. Приведем несколько величин, которые используют для оценки системы:

a) коэффициент сжатия

$$коэффициент\ сжатия = \frac{размер\ выходного\ файла\ (сжатые\ данные)}{размер\ входного\ файла\ (данные\ источника)}$$

Сжатие считается успешным, если выполняется условие $0 < коэффициент\ сжатия < 1$;

b) качество сжатия

качество сжатия = 100 * (1 – *коэффициент сжатия*);

c) скорость сжатия R, определяемая как отношение

$$R = \frac{k}{n}$$

и измеряемая в «количестве кодовых бит, приходящихся на отсчет данных источника»;

d) в случае использования необратимого сжатия необходимы метрики для измерения расхождения декодированных данных с исходными. Например, при сжатии изображений можно использовать величину искажений D:

$$D = \frac{1}{m}\sum_{i=1}^{n}\left(P_i - Q_i\right)^2,$$

P_i – пикселы исходного изображения, Q_i – пикселы восстановленного изображения (где $1 \le i \le m$).

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.

2. Универсальное сжатие

2.1. Неравномерное кодирование дискретных источников

Под дискретным источником будем понимать устройство, выдающее дискретную последовательность B, элементами которой являются a_i, принадлежащие некоторому алфавиту $A = \{a_1, \ldots, a_n\}$. Величина n называется объемом алфавита источника. Каждый элемент сообщения появляется на выходе источника с вероятностью $p(a_j)$, причем $\sum p(a_i) = 1$. Заметим, что рассматриваемый источник удобно использовать при побуквенном кодировании.

В основе методов неравномерного кодирования 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». Для однозначного декодирования неравномерного кода будем считать, что никакое кодовое обозначение не совпадает с началом какого-либо другого более длинного кодового обозначения (другими словами, код является префиксным).

Утверждение. Элемент a_i, вероятность появления которого равняется $p(a_i)$, выгоднее всего представлять – $\log_2 p(a_i)$ битами.■

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

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

это значение также называют энтропией источника в заданный момент времени.

Обычно вероятность появления элемента является условной. Тогда при кодировании очередного элемента a_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

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

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.

Студентам следует обратить внимание на тот факт, что статистические алгоритмы можно разделить на три класса:

- Неадаптивные – 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 в два этапа (на первом происходит подсчет частот, а на втором – кодирование).

➢ Адаптивные – 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.

Определение. Пусть при кодировании элементов (s_1, …, s_m) с вероятностями появления (p_1, …, p_m) получается код (c_1, …, c_m) с длинами кодовых слов (l_1, …, l_m), тогда средней длиной кодовых слов называется величина $l = \sum_{i=1}^{m} p_i l_i$.■

Определение. Избыточностью неравномерного кода называется величина $r = l – H$. Она показывает, что на каждое сообщение будет тратиться на r бит больше, чем потратилось бы, если использовать теоретически наилучший метод кодирования[1].■

Отметим, что неравномерные коды очень чувствительны к ошибкам в канале связи, т. к. полученные битовые последовательности упаковываются в байты без учета границ байтов, и при возникновении ошибок декодирование очередного символа может начаться с середины кода с выдачи бессмысленного результата.

2.1.1. Код Хаффмана

Статический алгоритм

Кодирование Хаффмана является простым алгоритмом для построения кодов переменной длины. Он весьма популярен и используется во многих методах сжатия текстовой и графической информации.

Одним из удобных способов описания двоичных кодов является их представление в виде двоичного кодового дерева. Для того чтобы построить m-е кодовое дерево для данного кода, начиная с некоторой точки – *корня* кодового дерева – будем проводить ветви с весами (0,1 …, m – 1). Будем считать, что

[1] Наилучшего метода кодирования программно может и не существовать.

на *листьях* кодового дерева находятся буквы алфавита источника, причем каждой букве будет соответствовать своя вершина и *свой путь от корня к вершине*. Кодом буквы будем считать запись, которая получается после применения операции конкатенации для весов ребер, входящих в путь от корня к листу.

Описание алгоритма приведено по работам [1; 2].

Исходными данными алгоритма являются:

- алфавит $A = \{a_1,\ldots, a_n\}$, состоящий из n символов;
- $p(a_i)$ – вероятность появления каждого символа алфавита в рассматриваемом сообщении.

Алгоритм Хаффмана для m-х кодов состоит из нескольких шагов.

Шаг 1. Инициализация:

1) добиваемся, чтобы число n можно было представить в виде $n = m + k(m – 1)$, где k – целое число. Этого можно добиться, добавив, если необходимо, к алфавиту A «фиктивные буквы» с нулевой вероятностью;

2) выстраиваем все символы текущего алфавита в порядке убывания вероятностей;

3) вводим вспомогательную переменную M – число необработанных узлов и список необработанных узлов S;

4) перед основным шагом алгоритма будем считать, что

- $M = n;$
- $S = A$.

Шаг 2. Основной этап работы алгоритма:

while M > 1

{

1) в списке S выбираем m элементов с наименьшими вероятностями;

2) исключаем эти узлы из списка S;

3) в список S вводим новый узел и приписываем ему суммарный вес исключенных узлов;

4) новый узел связать с исключенными узлами;

5) $M = M – (m – 1)$.

}

Алгоритм окончен.

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	$\frac{4}{10}$	$\frac{1}{10}$	$\frac{2}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$

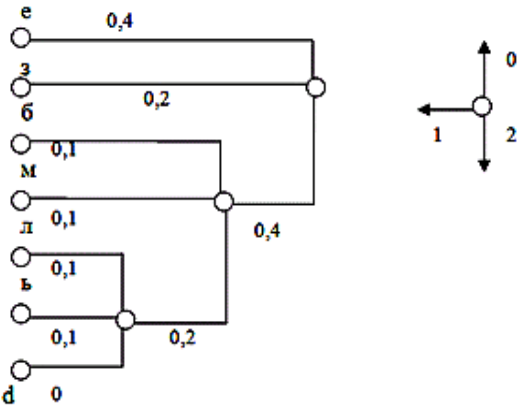
Закодировать алфавит с помощью троичного кода Хаффмана.

Решение.

Легко заметить, что равенство $6 = 3 + 2k$, где k – целое число, не выполняется. Присоединим в алфавит «фиктивный символ d» с нулевой вероятностью, и тогда равенство $7 = 3 + 2k$ выполняется. Следовательно, мы получили новый алфавит

символ	*е*	*б*	з	*м*	*л*	*ь*	*d*
вероятность	$\frac{4}{10}$	$\frac{1}{10}$	$\frac{2}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	0

Выполняем алгоритм Хаффмана для 3-го кода (рис. 2). Кодом буквы считаем запись, которая получается после применения операции конкатенации для весов ребер, входящих в путь от корня к листу.



символ	код
е	0
з	1
б	20
м	21
л	220
ь	221
d	222

Рис. 2. Кодовое дерево Хаффмана

Пример окончен.

Динамический (адаптивный) алгоритм

Основная идея динамического кодирования заключается в том, что кодер и декодер начинают работу с пустого дерева Хаффмана, а потом модифицируют его при обработке очередного символа. Описание алгоритма приведено по работе [3].

Заметим, что двоичное дерево Хаффмана должно обладать следующими свойствами:

1) внешние узлы (листья) кодового дерева имеют неотрицательный вес; каждый внутренний (родительский) узел имеет подчиненные (дочерние) узлы, а его вес равен сумме весов дочерних узлов;

2) на каждом уровне дерева (за исключением корневого) должно быть не менее одной пары узлов, имеющих общий родительский узел;

3) узлы могут быть перечислены в порядке возрастания таким образом, что узлы с номерами $(2j - 1)$ и $(2j)$ являются узлами одного уровня, а их общий родительский узел имеет более высокий уровень. Нумерация узлов соответствует тому порядку, в котором узлы объединяются в соответствии со статическим алгоритмом Хаффмана.

Приведем пример дерева, обладающего свойством соперничества (рис. 3).

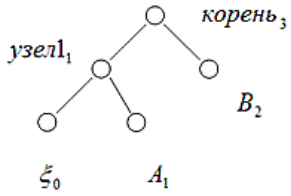


Рис. 3. Дерево Хаффмана

Дерево Хаффмана можно представить в виде упорядоченного списка.

Структура представления

Узел дерева	*Корень*	*B*	*узел 1*	*A*	*ξ*
Вес узла	3	2	1	1	0
Номер узла	5	4	3	2	1

Исходными данными алгоритма являются:

- алфавит $A = \{a_1, \ldots, a_n\}$, состоящий из n символов;
- строка $B = b_1 b_2 b_3 \ldots b_s$, которую надо закодировать и элементы, которой принадлежат алфавиту A.

Алгоритм Хаффмана состоит из нескольких шагов.

Введем обозначения:

- символ || обозначает операцию конкатенации;
- *код*(r) – функция, которая возвращает неравномерный код символа r, построенный по текущему двоичному дереву Хаффмана;
- *код*1(r) – функция, которая возвращает равномерный код символа r, построенный по исходному алфавиту, например 8-битовый код ASCII;
- // комментарии.

Шаг 1. Инициализация:

1) введем вспомогательный символ ξ с нулевой вероятностью, его кодовое значение в выходной строке будет сигнализировать, что следующим будет идти несжатый символ;

2) введем вспомогательную переменную *cod*;

3) инициализируем пустое двоичное дерево Хаффмана;

4) для каждого элемента алфавита $a_i \in A$ введем счетчик n_{a_i}. Отметим, что каждый узел двоичного дерева Хаффмана будет иметь вес. Пусть $n_{a_i} = 0$, $n_{\xi} = 0$.

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

{

1) считываем очередной символ b_j;

2) *If* ($n_{b_j} == 0$)

{

cod = *код*(ξ) || *код*1(n_{b_j})

выполнить процедуру *New*(b_j)

}

else cod = *код*(n_{b_j});

3) выдать в сжатый файл *cod*;

4) с помощью процедуры *Tree*(b_j) модифицируем текущее дерево, одновременно добиваясь, что бы оно являлось деревом Хаффмана[2]

}

Алгоритм окончен.

---

[2] При кодировании символа b_1 можно считать, что *cod* = *код*1(b_1)

Опишем процедуры *New*(*X*) и *Tree*(*X*) более подробно.

Процедура *New*(*X*) состоит из нескольких шагов:

1) создаем новый узел η с нулевым весом;

2) находим на текущем дереве лист символа ξ и заменяем его узлом η;

3) создаем новые листья с нулевыми весами ξ и X (родителем их является узел η). Пронумеруем узлы начиная с левого подчиненного узла: левый, правый, родительский.

// Процедура добавления нового символа состоит в том, что в хвост упорядоченного списка, коим является наше дерево кодов, добавляются два новых элемента. //

Процедура завершена.

Процедура *Tree*(*X*) состоит из двух шагов.

Шаг 1. Найдем на текущем дереве лист символа X (пусть мы нашли s – узел графа, n_s – вес узла).

Шаг 2. Работа алгоритма

If (*s and ξ являются дочерними узлами одного родителя*)
{
    Поменять местами *s* и лист, имеющий максимальный номер того же самого веса;
    $n_s = n_s + 1$;
    s – родитель s // новым текущим узлом становится родитель *s*
}
while (*true*)
{
    $n_s = n_s + 1$;
    If (*s корень*) *exit*;
    If ($n_s > n_{s+1}$)
    {
        $i = 1$;
        while ($n_s > n_{s+i}$) $i = i + 1$;
        // мы обнаружили группу, где может нарушаться упорядоченность. В этом случае следует поменять местами *s* и последний узел в этой группе (за исключением родителя); данное действие можно выполнять, используя упорядоченный список //

Перестановка_поддеревьев (*s, s + i*)[3];

}

s – родитель s // новым текущим узлом становится родитель *s*

}

Процедура завершена.

Пример. Дан алфавит *A* = {*а, в, о, р, т*}. Закодировать строку *B* = *ворота* с помощью динамического алгоритма Хаффмана.

Решение

Символы алфавита *A* могут быть закодированы равномерным кодом

а-(000) в-(001) о-(010) р-(110) т-(111)

Строим дерево Хаффмана (см. с. 14).

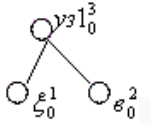
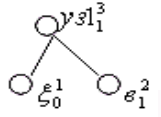
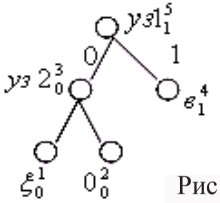
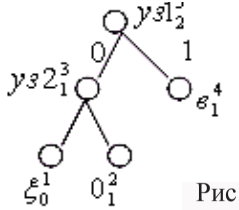
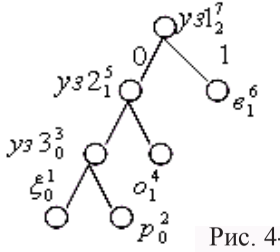
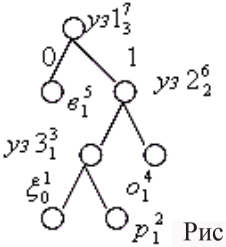
Студентам следует обратить внимание, что на рис. 4-ё (введение в дерево символа «р») нарушается свойство хаффманского дерева и оно нуждается в модификации. После модификации получим дерево с рис. 4-ж. Аналогично свойства хаффманского дерева нарушаются на рис.:4-з, 4-к, 4-м.

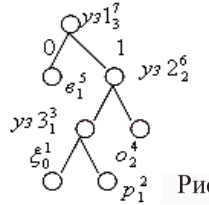
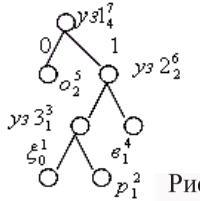
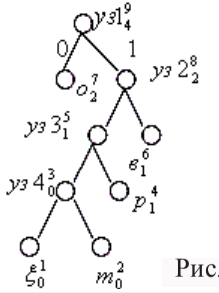
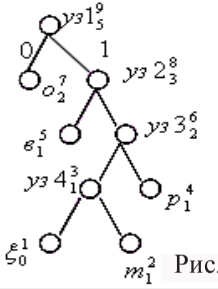
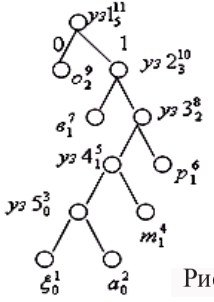
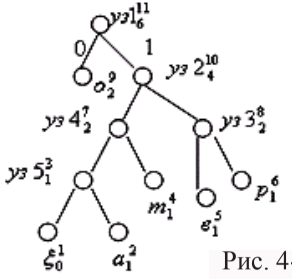
Учитывая биты, которые поступают на выход в процессе кодирования, получим, что слово «ворота» будет закодировано в последовательность 001001000110111001111000000.

Пример окончен.

[3] Поддерево *s* – это дерево, корнем которого является узел *s*.

Построение дерева Хаффмана *(к примеру со с. 13)*

Символ	*Дерево до преобразования*	*Представление*	*Дерево после преобразования*	*Выход*
инициа-лизация	 Рис. 4-а	Узел | ξ Вес | 0 номер | 1	 Рис. 4-б	
в	 Рис. 4-в	Уз1 | в | ξ 1 | 1 | 0 3 | 2 | 1	 Рис. 4-г	код1(в)
о	 Рис. 4-д	Уз1 | в | Уз2 | о | ξ 2 | 1 | 1 | 1 | 0 5 | 4 | 3 | 2 | 1	 Рис. 4-е	0код1(о)
р	 Рис. 4-ё	Уз1 | Уз2 | в | о | Уз3 3 | 2 | 1 | 1 | 1 7 | 6 | 5 | 4 | 3 р | ξ | | | | 1 | 0 | | | | 2 | 1 | | | |	 Рис. 4-ж	00код1(р)

о	 Рис. 4-з		 Рис. 4-и	11
т	 Рис. 4-к		 Рис. 4-л	100код1(т)
а	 Рис. 4-м		 Рис. 4-н	1000код1(а)

Уз1	Уз2	О	В	Уз3
4	2	2	1	1
7	6	5	4	3

p	ξ			
1	0			
2	1			

Уз1	Уз2	О	Уз3	В
5	3	2	2	1
9	8	7	6	5

p	Уз4	Т	ξ	
1	1	1	0	
4	3	2	1	

Уз1	Уз2	О	Уз3	Уз4
6	4	2	2	2
11	10	9	8	7

p	в	Т	Уз5	а
1	1	1	1	1
6	5	4	3	2

ξ
0
1

2.1.2. Код Шеннона

Описание алгоритма приведено по работе [2].

Исходными данными алгоритма являются:

- алфавит $A = \{a_1,\ldots, a_n\}$, состоящий из n символов;
- $p(a_i)$ – вероятность появления каждого символа алфавита в рассматриваемом сообщении.

Алгоритм Шеннона состоит из нескольких шагов.

Шаг 1. Инициализация:

1) выстраиваем все символы текущего алфавита в порядке убывания вероятностей. Будем считать, что выполняется условие $p_1 \geq \ldots \geq p_i \geq \ldots \geq p_n$;

2) для каждого a_i вводим вспомогательную переменную d_i;

3) будем считать, что $d_1 = 0$.

Шаг 2. Работа алгоритма:

1) *for* $i = 2$ *to* $|A|$ *do* $d_i = \sum_{k=1}^{i-1} p_k$;

2) в качестве кодового слова для каждого символа a_i берем первые $\lceil -\log p_i \rceil$ разрядов после запятой в двоичной записи числа d_i.

Алгоритм окончен.

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

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

Закодировать алфавит с помощью алгоритма Шеннона.

Решение

Процесс работы алгоритма можно описать следующей таблицей.

символ	*е*	з	*б*	*м*	*л*	*ь*
вероятность	$\frac{4}{10}$	$\frac{2}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$
d_i	0	0,4	0,6	0,7	0,8	0,9
$\lceil -\log p_i \rceil$	2	3	4	4	4	4
кодовое слово	00	011	1001	1011	1100	1110

Следовательно, слово «безземелье» будет закодировано в последовательность 10010001101100101100110011100 0

Пример окончен.

2.1.3. Код Гилберта–Мура

Описание алгоритма приведено по работе [2].

Исходными данными алгоритма являются:

- алфавит $A = \{a_1,\ldots, a_n\}$, состоящий из n символов;
- $p(a_i)$ – вероятность появления каждого символа a_i в сообщении.

Алгоритм Гилберта–Мура состоит из нескольких шагов.

Шаг 1. Инициализация.

1) для каждого элемента алфавита a_i введем вспомогательные переменные d_i, δ_i;

2) будем считать $d_1 = 0$, $\delta_1 = \frac{p_1}{2}$.

Шаг 2. Работа алгоритма.

1) *for* $i = 2$ *to* $|A|$ *do*

{

$$d_i = \sum_{k=1}^{i-1} p_k ;$$

$$\delta_i = d_i + \frac{p_i}{2}$$

}

2) в качестве кодового слова для каждого символа a_i берем первые $\lceil -\log p_i \rceil + 1$ разрядов после запятой в двоичной записи числа δ_i.

Алгоритм окончен.

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

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

Закодировать алфавит с помощью алгоритма Гилберта–Мура.

Решение

Процесс работы алгоритма можно описать следующей таблицей.

символ	*б*	*е*	з	*м*	*л*	*ь*
вероятность	0,1	0,4	0,2	0,1	0,1	0,1
d_i	0	0,1	0,5	0,7	0,8	0,9
δ_i	0,05	0,3	0,6	0,75	0,85	0,95
$\lceil -\log p_i \rceil + 1$	5	3	4	5	5	5
кодовое слово	00001	010	1001	11000	11011	11110

Пример окончен.

2.2. Словарные методы

Идея словарных методов:

- рассматриваем входную последовательность как последовательность строк, содержащих произвольное количество символов;
- при кодировании заменяем строки символов на кода (индексы строк в некотором словаре);
- при декодировании осуществляется замена индексов на соответствующие им фразы словаря.

Словарь – это набор строк, которые предположительно будут встречаться в обрабатываемой последовательности. Индексы строк должны быть построены таким образом, чтобы в среднем их представление занимало меньше места, чем требуют замещаемые строки. За счет этого и происходит сжатие.

2.2.1. Алгоритм LZW

Впервые алгоритм был опубликован в 1984 г. Он применяется, например, для сжатия данных при записи их на жесткий диск компьютера в форматах ARJ, ZIP; а также при записи файлов изображений в форматах TIFF и GIF.

При кодировании вводится поток символов и выводится поток кодов. Перед описанием алгоритма кодирования введем несколько обозначений и определений:

K – символ простейший элемент данных (в тексте – это может быть элемент алфавита, а при рассмотрении изображений это может быть числовое значение цвета в палитре);

w – строка, несколько непрерывных символов от одного и более;

wK – операция конкатенации строки и символа;

код – число, помещаемое в выходной файл.

Алгоритм кодирования состоит из нескольких шагов.

Шаг 1. Инициализация словаря всеми возможными односимвольными фразами. Инициализация входной фразы *w* первым символом сообщения.

Шаг 2. Считать очередной символ *K* из кодируемого сообщения.

Шаг 3. Если конец_сообщения,

Выдать код для *w*.

Конец.

Если фраза *wK* уже есть в словаре,

Присвоить входной фразе значение *wK*, то есть *w* = *wK*.

Перейти к шагу 2.

Иначе

Выдать код *w*

Добавить *wK* в словарь

Присвоить входной фразе значение *K*, то есть *w* = *K*.

Перейти к шагу 2.

Алгоритм окончен.

При декодировании вводится поток кодов и выводится поток символов. Перед описанием алгоритма декодирования введем несколько обозначений и определений:

> *строка* (*b*) – функция, которая по коду *b* выдает из словаря связанную с ним строку.

> *строка′* (*b*) – функция, которая по коду *b* выдает первый символ из связанной с ним строки.

Алгоритм декодирования состоит из нескольких шагов.

Шаг 1. Инициализация.

Инициализация словаря всеми возможными односимвольными фразами.

Ввести переменные w – текущий код, s – старый код, t – строка символов и K – символ.

Считать w – первый код из сжатого файла.

Вывести *строку* (w) в поток символов.

$s = w$.

Шаг 2. Работа алгоритма:

w = *следующий код в потоке кодов*.

Если код w существует в словаре

{

вывести *строка* (w) в поток символов;

t = *строка* (s);

K = *строка'* (w);

добавить фразу tK в словарь;

$s = w$.

}

Иначе

{

t = *строка* (s);

K = *строка'* (s);

вывести tK в поток символов;

добавить фразу tK в словарь;

$s = w$.

}

Повторить шаг 2 нужное количество раз.

Алгоритм окончен.

Пример. Рассмотрим работу алгоритма LZW с помощью кодирования фразы «abcdabce_abcd_abceabcabcabcd». Размер словаря не ограничен.

Решение

Процесс кодирования можно описать таблицей.

	вход	*текущая фраза wK*	*фраза, помещаемая (в словарь)*	*позиция в словаре*	*комментарии*	*коды на выходе*
1			_ a b c d e	0 1 2 3 4 5	Инициализация словаря всеми возможными односимвольными фразами	
2	a					
3	b	ab	ab	6	фрагмента «ab» нет в словаре; фрагмент «a» есть в словаре	1
4	c	bc	bc	7	фрагмента «bc» нет в словаре	2
5	d	cd	cd	8	-//-	3
6	a	da	da	9	-//-	4
7	b	ab			фрагмент «ab» есть в словаре	
8	c	abc	abc	10	фрагмента «abc» нет в словаре	6
9	e	ce	ce	11	фрагмента «ce» нет в словаре	3
10	_	e_	e_	12	-//-	5
11	a	_a	_a	13	-//-	0
12	b	ab			фрагмент «ab» есть в словаре	
13	c	abc			фрагмент «abc» есть в словаре	
14	d	abcd	abcd	14	фрагмента «abcd» нет в словаре	10
15	_	d_	d_	15	-//-	4
16	a	_a			фрагмент «_a» есть в словаре	
17	b	_ab	_ab	16	фрагмента «_ab» нет в словаре	13
18	c	bc			фрагмент «bc» есть в словаре	
19	e	bce	bce	17	фрагмента «bce» нет в словаре	7
20	a	ea	ea	18	фрагмента «ea» в словаре нет	5
21	b	ab			фрагмент «ab» есть в словаре	
22	c	abc			фрагмент «abc» есть в словаре	

23	a	abca	abca	19	фрагмента «abca» в словаре нет	10
24	b	ab			фрагмент «ab» есть в словаре	
25	c	abc			фрагмент «abc» есть в словаре	
26	a	abca			фрагмент «abca» есть в словаре	
27	b	abcab	abcab	20	фрагмента «abcab» в словаре нет	19
28	c	bc			фрагмент «bc» есть в словаре	
29	d	bcd	bcd	21	фрагмента «bcd» в словаре нет	7
30		d			получили конец файла выдали код строки	4

Таким образом на выходе мы получим строку (1, 2, 3, 4, 6, 3, 5, 0, 10, 4, 13, 7, 5, 10, 19, 7, 4)

Процесс декодирования можно описать таблицей.

	вход кода w	*текущая фраза*	*фраза, помещаемая (в словарь)*	*позиция в словаре*	*комментарии*	*выход*
1			_ a b c d e	0 1 2 3 4 5	Инициализация словаря всеми возможными односимвольными фразами	
2	1	a			код 1 есть в словаре выводим в поток «a» $s = 1$	a
3	2	b	ab	6	код 2 есть в словаре, выводим в поток «b», $s = 1$, $t = строка(s) = $ «a», $K = строка'(w) = $ «b», помещаем фразу $tK = $ «ab» в словарь, $s = w = 2$	b
4	3	c	bc	7	код 3 есть в словаре, выводим в поток «c», помещаем фразу «bc» в словарь	c
5	4	d	cd	8	код 4 есть в словаре, выводим в поток «d», помещаем фразу «cd» в словарь	d

6	6	ab	da	9	код 6 есть в словаре, выводим в поток «ab», $s = 4$, $t =$ «*d*», $K =$ «a», помещаем фразу «da» в словарь, $s = 6$	ab
7	3	c	abc	10	код 3 есть в словаре, выводим в поток «c», $s = 6$, $t =$ «ab», $K =$ «c», помещаем фразу «abc» в словарь, $s = 3$	c
8	5	e	ce	11	код 5 есть в словаре, помещаем фразу «ce» в словарь.	e
9	0	_	e_	12	код 0 есть в словаре, помещаем фразу «e_» в словарь	_
10	10	abc	_a	13	код 10 есть в словаре, выводим в поток «abc», $s = 0$, $t =$ «_», $K =$ «a», помещаем фразу «_a» в словарь, $s = 10$	abc
11	4	d	abcd	14	код 4 есть в словаре, помещаем фразу «abcd» в словарь	d
12	13	_a	d_	15	код 13 есть в словаре, помещаем фразу «d_» в словарь	_a
13	7	bc	_ab	16	код 7 есть в словаре, помещаем фразу «_ab» в словарь	bc
14	5	e	bce	17	код 5 есть в словаре, помещаем фразу «bce» в словарь	e
15	10	abc	ea	18	код 10 есть в словаре, $s = 5$, $t =$ «*e*», $K =$ «a», помещаем фразу «ea» в словарь. $s = 10$	abc
16	19		abca	19	кода19 нет в словаре, $s = 10$, $t =$ *строка* (s) = «abc», $K =$ *строка'* (s) = «a», вывести $tK =$ «abca» в поток символов, добавить фразу $tK =$ «abca» в словарь, $s = w = 19$	abca
17	7	bc	abcab	20	код 7 есть в словаре, $s = 19$, $t =$ «abca», $K =$ «b», помещаем фразу «abcab» в словарь. $s = 7$	bc
18	4	d	bcd	21	код 4 есть в словаре, помещаем фразу «bcd» в словарь	d

Таким образом, на выходе мы получим фразу «abcdabce_ abcd_abceabcabcabcd»

Пример окончен.

2.3. Монотонные коды

Определение. Монотонным кодом назовем префиксный код[4] множества натуральных чисел, который удовлетворяет следующему условию: если для любых $i, j \in N$ и $i \leq j$ то для длин соответствующих кодовых слов l_i и l_j выполняется неравенство $l_i \leq l_j$ ■

Монотонные коды можно использовать, например для кодирования индексов строк словаря в алгоритме LZW.

2.3.1. Унарный код

Унарный код сопоставляет натуральному числу m двоичную комбинацию вида $1^{m-1}0$ [5], альтернативной записью унарного кода является $0^{m-1}1$ [9, 2].

Приведем несколько унарных кодов.

m	унарный код *unar*	альтернативный унарный код	m	унарный код	альтернативный унарный код
1	0	1	5	11110	00001
2	10	01	6	111110	000001
3	110	001	7	1111110	0000001
4	1110	0001	8	11111110	00000001

Легко заметить, что длина кодового слова числа i для унарного кода равна $l_i = i$.

2.3.2. Код Голомба

Код является параметризированным префиксным кодом. Код особенно полезен в тех случая, когда удается подобрать хорошие значения параметра T. Код Голомба для натурального числа m

---

[4] Напомним, что кодирование f называется префиксным, если никакое кодовое слово $f(a_i)$ не является префиксом (началом) другого кодового слова $f(a_j)$. Примером префиксного кода является код Хаффмана.

[5] Запись вида 1^m или 0^m означает последовательность из m единиц или нулей соответственно.

и параметра T [6] состоит из двух частей и может быть получен после выполнения следующих шагов [12, 2]:

- вычисляем величины $q = \left\lfloor \frac{m}{T} \right\rfloor$, $r = m - qT$ и $b = \lceil \log_2 T \rceil$;
- строим первую часть кода (кодируем число $q + 1$ с помощью унарного кода);
- второй частью кода является двоичное представление числа r:

– если $r < 2^b - T$, то двоичная запись числа r размещается в b –1 битах;

– остальные числа кодируются в b битах.

Код Голомба для натурального числа m и параметра T [7] состоит из двух частей и может быть получен после выполнения следующих шагов:

- вычисляем величины $q = \left\lfloor \frac{m}{T} \right\rfloor$ и $r = m - qT$;
- строим первую часть кода (кодируем число $q + 1$ с помощью унарного кода);
- второй частью кода является двоичное представление числа r, состоящее из b бит.

Приведем несколько кодов Голомба.

T \ m	1	2	3	4	5	6	7	8	9
2	(0)1	(10)0	(10)1	(110)0	(110)1	(1110)0	(1110)1	(11110)0	(11110)1
3	(0)10	(0)11	(10)0	(10)10	(10)11	(110)0	(110)10	(110)11	(1110)0
4	(0)01	(0)10	(0)11	(10)00	(10)01	(10)10	(10)11	(110)00	(110)01
5	(0)01	(0)10	(0)110	(0)111	(10)00	(10)01	(10)10	(10)110	(10)111
6	(0)01	(0)100	(0)101	(0)110	(0)111	(10)00	(10)01	(10)100	(10)101

Пусть входной поток данных состоит из натуральных чисел m и вероятность числа равна $P(m) = (1-p)^{m-1}$, (где p – некоторый параметр $0 \le p \le 1$). Коды Голомба будут оптимальными кодами для этого потока данных, если T выбрать из условия $(1-p)^T + (1-p)^{T+1} \le 1 < (1-p)^{T-1} + (1-p)^T$.

Легко заметить, что длина кодового слова числа i для кода Голомба равна $l_i = \left\lfloor \frac{i}{T} \right\rfloor + 1 + \lceil \log_2 T \rceil$.

[6] Пусть T не является степенью 2.

[7] Пусть T является степенью 2, то есть $T = 2^b$.

2.3.3. Код Левенштейна

Введем обозначения:

- $bin(i)$ – двоичная запись натурального числа i;
- $bin'(i)$ – двоичная запись натурального числа i без первой единицы;
- $|\, bin(i)\,|$ – длина двоичной последовательности $bin(i)$;
- $\lambda\,|\, bin'(i)\,|$ – длина двоичной последовательности $bin'(i)$.

Студенты могут заметить, что непосредственно использовать двоичные представления натуральных чисел при кодировании нельзя, ибо такой код не будет префиксным. Например, $bin(2) = 10$ является префиксом $bin(5) = 101$. Простейший способ преобразования двоичной записи числа в префиксный код состоит в том, что в начало можно записать префикс, например указывающий на длину двоичной записи числа. Учитывая, что в двоич-ной записи старший разряд всегда 1, его можно не передавать, если декодер знает длину двоичной записи (следовательно, число i можно, например, закодировать $\left(unar\left(|\, bin'(i)| + 1\right), bin'(i)\right)$).

Чтобы сделать запись еще короче, с длиной двоичной записи можно поступить так же, как и с самим числом, то есть заменим $|\, bin'(i)\,|$ на $\left|\, bin'\,|\, bin'(i)\,\right|$, $bin'\,|\, bin'(i)\,|$ и так далее[8]. Итерации продолжаются, пока не останется один значащий разряд. Чтобы декодирование было однозначным, достаточно приписать префикс, содержащий закодированное префиксным кодом число итераций. Заметим, что минимальное число итераций равно 0 (при кодировании числа 1). Поэтому в качестве префиксного кода можно выбрать унарный код числа t (где t равно увеличенному на единицу числу итераций). Полученное кодовое слово будет кодовым словом кода Левенштейна.

Пример: построим код Левенштейна для числа $m = 17$.

Решение

$bin(17) = 10001 \Rightarrow bin'(17) = 0001 \Rightarrow |\, bin'(17)\,| = 4$;

$bin(4) = 100 \Rightarrow bin'(4) = 00 \Rightarrow |\, bin'(4)\,| = 2$;

$bin(2) = 10 \Rightarrow bin'(2) = 0 \Rightarrow |\, bin'(2)\,| = 1$.

---

[8] Другими словами, $|\, bin'(i)\,|$ можно представить $bin'(\lambda^{k-2}(i))\ldots bin'(\lambda(i))$, где $\lambda^2(i) = |\, bin'((\lambda(i))\,|$ и так далее.

Число итераций равно 3, следовательно в качестве префикса можно взять *unar*(4) = 1110.

lev(17) = (1110)(0)(00)(0001).

Декодер кода Левенштейна сначала узнает, что число итераций равно 3. Прочитав значащий разряд (0) и дописав к нему в начало 1, получаем последовательность 10, то есть на предпоследней итерации длина числа была 2. Прочитав два разряда и дописав слева 1, получим 100. Теперь считываем четыре разряда и дописываем слева 1. Получаем последовательность 10001, ей соответствует число $m = 17$.

Пример окончен.

Приведем несколько кодов Левенштейна.

Введем обозначения

1 – $bin\,(m)$ 4 – $bin\left(|\,bin'\,(m)|\right)|$ 7 – $bin\left(\left|\,bin'\left(|\,bin'\,(m)|\right)\right|\right)$

2 – $bin'\,(m)$ 5 – $bin'\left(|\,bin'\,(m)|\right)$ 8 – $bin'\left(\left|\,bin'\left(|\,bin'\,(m)|\right)\right|\right)$

3 – $|\,bin'\,(m)|$ 6 – $\left|\,bin'\left(|\,bin'\,(m)|\right)\right|$ 9 – $\left|\,bin'\left(\left|\,bin'\left(|\,bin'\,(m)|\right)\right|\right)\right|$

m	1	2	3	4	5	6	7	8	9	число итераций	*lev*
21	10101	0101	4	100	00	2	10	0	1	3	(1110)(0)(00)(0101)
25	11001	1001	4	100	00	2	10	0	1	3	(1110)(0)(00)(1001)
30	11110	1110	4	100	00	2	10	0	1	3	(1110)(0)(00)(1110)
40	101000	01000	5	101	01	2	10	0	1	3	(1110)(0)(01)(01000)
60	111100	11100	5	101	01	2	10	0	1	3	(1110)(0)(01)(11100)
80	1010000	010000	6	110	10	2	10	0	1	3	(1110)(0)(10)(010000)

Другим похожим кодом может служить монотонный код

$$mon(i) = \left(unar\left(|\,bin'(i)\,| + 1\right),\, bin'(i)\right).$$

Студенты могут легко заметить, что согласно определению $|\,bin'(i)\,| = \lfloor \log i \rfloor$ и значит длинна кодового слова числа i (монотонный код) равна $l_i = 2\lfloor \log i \rfloor + 1$.

2.3.4. Код Элайеса

Код является упрощенным кодом Левенштейна. Числу $m = 1$ припишем кодовое слово $elias(1) = 0$. Для чисел $m > 1$ кодовые слова вычисляются по формуле:

$$elias(m) = \left(unar\left(\left|\, bin'\left(\left|\, bin'(m)\right|\right)\right| + 2\right),\ bin'\left(\left|\, bin'(m)\right|\right),\ bin'(m)\right).$$

Приведем несколько кодов Элайеса.

m	$bin(m)$	$s = bin'(m)$	$\lvert s\rvert$	$bin(\lvert s\rvert)$	$bin'(\lvert s\rvert)$	$s_1 = \lvert bin'(\lvert s\rvert)\rvert$	$unar(s_1 + 2)$	$elias(m)$
21	10101	0101	4	100	00	2	1110	(1110)(00)(0101)
25	11001	1001	4	100	00	2	1110	(1110)(00)(1001)
30	11110	1110	4	100	00	2	1110	(1110)(00)(1110)
40	101000	01000	5	101	01	2	1110	(1110)(01)(01000)
60	111100	11100	5	101	01	2	1110	(1110)(01)(11100)
80	1010000	010000	6	110	10	2	1110	(1110)(10)(010000)

Длинна кодового слова числа i для кода Элайеса равна

$$l_i = \begin{cases} 1, & i = 1 \\ \lfloor \log i \rfloor + 2\lfloor \log \lfloor \log i \rfloor \rfloor + 2 & i > 1 \end{cases}$$

3. Сжатие изображений

3.1. Введение

Определение. Цифровое изображение представляет собой прямоугольную таблицу (матрицу) точек, размер которой $n \times m$, где n – число столбцов, m – число строк в изображении. Будем считать, что для каждой точки (x, y) определена положительная скалярная величина $f(x, y)$ – отсчет, физический смысл которого определяется источником изображения. ■

Принцип сжатия изображений. *Если случайно выбрать пиксель изображения, то с большой вероятностью ближайшие к нему пиксели будут иметь тот же или близкий цвет. Другими словами, мы можем ожидать, что ближайшие пиксели изображения коррелируют между собой.*

Студенты должны обратить внимание на то, что для корректной оценки степени сжатия изображения и целесообразности использования того или иного алгоритма сжатия к данному изображению следует учитывать класс изображения. Под классом цифрового изображения понимается совокупность изображений, которая при применении к ней выбранного алгоритма сжатия дает качественно одинаковый результат.

Традиционными классами изображений являются следующие.

• ***Монохроматическое изображение***. В этом случае все пикселы могут иметь только два значения и значит каждый пиксел изображения представлен одним битом;

• ***Цветное изображение***. Все цветные изображения можно подразделить на две группы: с палитрой и без неё. Для изображений с палитрой $f(x, y)$ – это индекс в некотором одномерном векторе цветов, называемом палитрой. Палитры обычно бывают 8, 16 и 256 элементными. Изображения без палитры обычно бывают в определенной системе цветопредставления. Существует несколько методов задания цвета в цветовой модели, но обычно в описании участвуют три параметра. Как правило, цветной пиксель состоит из трех байтов. Типичными цветовыми моделями являются: *RGB*, *CMY*, *HSV*, YC_rC_b. Опишем указанные цветовые модели подробнее.

– *RGB* (модель является основной при создании и обработке компьютерной графики, предназначенной для электронного воспроизведения).

Модель *RGB* является аддитивной, то есть любой цвет представляется сочетанием в различных пропорциях трех основных цветов – красного, зеленого, синего (рис. 5).

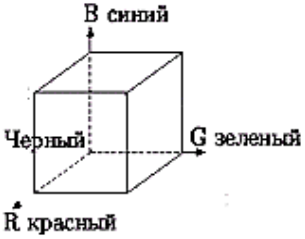


Рис. 5. Модель RGB

– *CMY* (модель используют при подготовке публикаций к печати. Модель описывается тремя компонентами: голубым, пурпурным, желтым).

Цвета *CMY* описывают отраженный от белой бумаги свет трех основных цветов RGB модели (рис. 6).

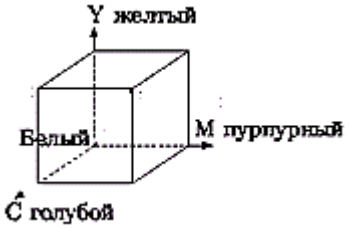


Рис. 6. Модель CMY

Соотношения для перекодировки цвета из модели *CMY* в *RGB:*

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

И обратно – из модели *RGB* в *CMY:*

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Здесь считается, что компоненты кодируются числами в диапазоне от 0 до 1. Для иного диапазона чисел можно записать соответствующие соотношения.

– ***HSV*** (цвет описывается тремя компонентами: цветовым тоном, насыщенностью, светлотой) [4];

Цветовой тон ***H*** измеряется в градусах от 0 до 60, цвета располагаются по кругу в таком порядке — красный, оранжевый, желтый, зеленый, голубой, синий и т. д. (рис. 7).

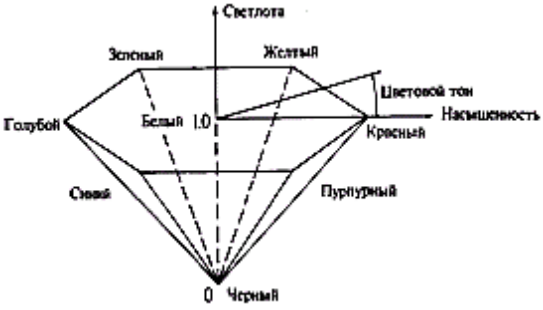


Рис. 7. Модель HSV

Значения насыщенность S и светлота V находятся в диапазоне (0...1). Отметим, что насыщенность зависит от цветового охвата.

Алгоритм преобразования цветового пространства HSV в RGB

Исходными данными алгоритма являются*:*

- *H* цветовой тон (0–360˚) 0˚ – красный.
- *S* насыщенность (0–1).
- *V* светлота (0–1).

Выходными данными алгоритма являются *RGB* – красный, зеленый, синий; основные цвета (0–1).

Алгоритм преобразования

Пусть *Floor* – функция, выделяющая целую часть числа.

```
If S = 0 then
{
    If H = неопределенность then
    {
        R = V;
        G = V;
        B = V;
    }
    else если H определено, то ошибка
}
```

else

{

If H = 360 *then H* = 0;

else H = *H* / 60;

I = *Floor*(*H*);

F = *H* – *I*;

M = *V* * (1 – *S*);

N = *V* * (1 – *S* * *F*);

K = *V* * (1 – *S* * (1 – *F*));

If I = 1 *then* (*R, G, B*) = (*V, K, M*)[9]

If I = 2 *then* (*R, G, B*) = (*N, V, M*)

If I = 3 *then* (*R, G, B*) = (*M, V, K*)

If I = 4 *then* (*R, G, B*) = (*M, N, V*)

If I = 5 *then* (*R, G, B*) = (*K, M, V*)

If I = 6 *then* (*R, G, B*) = (*V, M, N*)

}

Алгоритм окончен.

Алгоритм преобразования цветового пространства RGB в HSV

Исходными данными алгоритма являются *RGB* – красный, зеленый, синий; основные цвета (0–1).

Выходными данными алгоритма являются*:*

- *H* цветовой тон (0–360°) 0° – красный.
- *S* насыщенность (0–1).
- *V* светлота (0–1).

Алгоритм преобразования

Пусть *Max*, *Min* – функции определения максимума и минимума.

V = *Max* (*R, G, B*);

Temp = *Min* (*R, G, B*);

If V = 0 *then S* = 0;

else S = (*V* – *Temp*) / *V*;

If S = 0 *then H* = *неопределенность*;

else

[9] (*R, G, B*) = (*V, K, M*) означает *R* = *V, G* = *K, B* = *M*.

{

$C_r = (V - R) / (V - Temp)$;

$C_g = (V - G) / (V - Temp)$;

$C_b = (V - B) / (V - Temp)$;

If $R = V$ *then* $H = C_b - C_g$;

If $G = V$ *then* $H = 2 + C_r - C_b$;

If $B = V$ *then* $H = 4 + C_g - C_r$;

$H = 60 * H$;

If $H < 0$ *then* $H = H + 360$;

}

Алгоритм окончен.

– *YC$_r$C$_b$* (модель описывается тремя компонентами: Y – яркость, C_r – хроматический красный, C_b – хроматический синий). В отличие от *RGB*, где все компоненты равнозначны, цветовая модель YC_rC_b концентрирует наиболее важную информацию в одном из компонентов. Данная модель применима к сжатию за счет того, что человеческий глаз менее чувствителен к цвету, чем к яркости. Упрощенно перевод из цветового пространства *RGB* в цветовое пространство YC_rC_b можно представить с помощью матрицы перехода (опишем то, которое используется в формате jpeg [5]) :

$$\begin{bmatrix} Y \\ C_r \\ C_b \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}5 & -0{,}4187 & -0{,}0813 \\ -0{,}1687 & -0{,}3313 & 0{,}5 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

Обратное преобразование:

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

• ***Полутоновое изображение***. Каждый пиксель такого изображения может иметь значение из диапазона от 0 до $2^S - 1$, обозначающее одну из 2^S градаций серого (или иного) цвета. Число s обычно сравнимо с размером байта, то есть равно 4, 8, 12, 16 и так далее. Каждая градация серого цвета определяется ярко-

стью *Y*. Серые цвета в модели RGB описываются одинаковыми значениями компонентов, то есть $R = G = B$. Для преобразования цветного изображения, представленного в системе RGB, в градации серого можно воспользоваться соотношением:

$$Y = 0{,}299R + 0{,}587G + 0{,}114B,$$

где коэффициенты при *R, G, B* учитывают различную чувствительность зрения к соответствующим цветам.

• ***Изображение с непрерывным тоном***. Этот тип изображений может иметь много похожих цветов (или полутонов). Когда соседние пиксели отличаются всего на единицу, глазу практически невозможно различить их цвета.

Можно выделить и специфические классы изображений, такие как рентгеновские снимки, томаграфические изображения, радиолокационные планы местности и т. д.

Методы сжатия изображений обычно разрабатываются для конкретного класса изображений. Например, для монохроматического изображения можно эффективно использовать метод кодирования длин серий (RLE). Предполагаем, что отсчеты $f(x_i, y_j)$ непосредственных соседей пиксела (x, y) совпадают с $f(x, y)$. Метод сжатия RLE упрощенно можно описать следующим образом:

- изображение вытягивают в цепочку бит по строкам растра;
- находят числа, соответствующие длинам участков, на которых данные сохраняют неизменное значение;
- длины кодируют кодами переменной длины и записываются в сжатый файл.

3.2. Дискретные преобразования

Сжатие изображения достигается, если в процессе какого-либо преобразования получаются значения, которые имеют существенно меньшую корреляцию между собой, чем исходные отсчеты. Рассмотрим несколько известных дискретных преобразований, которые можно использовать для сжатия изображения.

Дискретное косинусное преобразование (DCT)

Дискретным косинусным преобразованием вектора $X = (x_0, x_1, \ldots, x_{n-1})$ назовем вектор $Y = (y_0, y_1, \ldots, y_{n-1})$, компоненты которого определяются по формуле:

$$y_k = \sqrt{\frac{2}{n}} \cdot C_k \sum_{t=0}^{n-1} x_t \cos\left(\frac{(2t+1)k\pi}{2n}\right), \text{ где } C_k = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & k > 0 \end{cases}, \ 0 \le k \le n-1.$$

Тогда обратное дискретным косинусным преобразованием имеет вид:

$$x_t = \sqrt{\frac{2}{n}} \sum_{k=0}^{n-1} C_k y_k \cos\left(\frac{(2t+1)k\pi}{2n}\right), \text{ где } C_k = \begin{cases} 1/\sqrt{2}, & k = 0 \\ 1, & k > 0 \end{cases}, \ 0 \le t \le n-1.$$

Двумерное преобразование (DCT) оперирует блоками P размером $n \times n$, элементами которых служат сэмплы (обычно это отсчеты самого изображения или величины разностей между отсчетом и прогнозом изображения), и порождает блок $n \times n$ некоторых коэффициентов G. Действие DCT (а также обратного к нему преобразования IDCT) можно описать с помощью матрицы преобразования M.

Прямое преобразование будет выглядеть как $G = MPM^T$ обратное $P = M^T GM$. Элементы матрицы M равны:

$$M_{ij} = C_i \cos\left(\frac{(2j+1)i\pi}{2n}\right), \text{ где } C_i = \begin{cases} \sqrt{\frac{1}{n}}, & i = 0 \\ \sqrt{\frac{2}{n}}, & i > 0 \end{cases}, \ 0 \le i, j \le n-1.$$

В случае если изображение разбивается на блоки пикселей p_{xy} размера 8×8, уравнение, используемое для нахождения коэффициентов G_{ij} может быть записано следующим образом:

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

где $C_k = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & k > 0 \end{cases}, \ 0 \le i, j \le 7.$

Обратное преобразование DCT вычисляется по формуле

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

где $\tilde{N}_k^{C_k} = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & k > 0 \end{cases}$, $0 \le i, j \le 7$.

Пример. Найти матрицу дискретного косинусного преобразования, если $n = 4$.

Решение

Матрица преобразования M имеет вид:

$$M = \begin{pmatrix} \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{5\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{7\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{2\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{6\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{10\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{14\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{9\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{15\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{21\pi}{8}\right) \end{pmatrix}$$

Можно легко вычислить, что

$$M = \begin{pmatrix} 0{,}5 & 0{,}5 & 0{,}5 & 0{,}5 \\ 0{,}653 & 0{,}271 & -0{,}271 & -0{,}653 \\ 0{,}5 & -0{,}5 & -0{,}5 & 0{,}5 \\ 0{,}271 & -0{,}653 & 0{,}653 & -0{,}271 \end{pmatrix}$$

Пример окончен.

Дискретное псевдокосинусное преобразование

Рассмотрим более простую в вычислении альтернативу *DCT* [8]. Алгоритм сжатия изображений на основе такого преобразования может быть востребован в устройствах с ограниченными аппаратными ресурсами.

Рассмотрим матрицы W_2, W_3, W_4, которые будем называть *элементарными матрицами дискретного псевдокосинусного преобразования (ДПКП)* .

$$W_2 = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad W_3 = \begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{2}} & 0 & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{6}} & -\sqrt{\frac{2}{3}} & \frac{1}{\sqrt{6}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{3}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{6}} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -2 & 1 \end{pmatrix}$$

$$W_4 = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{10}} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{10}} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{2}{\sqrt{10}} & \frac{1}{\sqrt{10}} & -\frac{1}{\sqrt{10}} & -\frac{2}{\sqrt{10}} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{\sqrt{10}} & -\frac{2}{\sqrt{10}} & \frac{2}{\sqrt{10}} & -\frac{1}{\sqrt{10}} \end{pmatrix}$$

Заметим, что рассмотренные матрицы могут быть представимы в виде:

$$W_n = D_n C_n,$$

где D_n – диагональная матрица нормирующих множителей, C_n – матрица, состоящая только из чисел $\{-2,-1,0,1,2\}$.

Легко убедиться, что матрицы W_2, W_3, W_4 ортогональны, поэтому $W_i^{-1} = W_i^T$, где $(2 \le i \le 4)$.

Определение. Кронекеровским произведением матриц A и B называется матрица C, образуемая по следующему правилу: j-й элемент матрицы C представляет собой соответствующий элемент a_{ij} матрицы A, умноженный на всю матрицу B, то есть $c_{ij} = a_{ij}B$. Для кронекеровского произведения матриц примем обозначение $C = A \otimes B$. Кронекеровское произведение матриц обладает следующими свойствами:

$$(A+B)\otimes C = A\otimes C + B\otimes C; \quad A\otimes(B+C) = A\otimes B + A\otimes C;$$

$$(A_1 A_2 \dots A_l)\otimes(B_1 B_2 \dots B_l) = (A_1\otimes B_1)(A_2\otimes B_2)\dots(A_l\otimes B_l);$$

$$(A_1\otimes A_2\otimes\dots\otimes A_l)(B_1\otimes B_2\otimes\dots\otimes B_l) = (A_1 B_1)\otimes(A_2 B_2)\otimes\dots\otimes(A_l B_l). \blacksquare$$

Матрицы дискретного псевдокосинусного преобразования размерности $n > 4$ предлагается строить следующим образом:

$$W_6 = W_2 \otimes W_3, \; W_6 = W_2 \otimes W_3, \; W_6 = W_2 \otimes W_3, \; W_6 = W_2 \otimes W_3, \; W_6 = W_2 \otimes W_3.$$

Другими словами число n должно быть представимо в виде произведения чисел 2, 3, 4. Располагать матрицы W_2, W_3, W_4 следует в порядке увеличения нижнего индекса.

Пусть $W_n = D_n C_n$, где $n = n_1, n_2, \ldots, n_s$, $n_i \in \{2, 3, 4\}$, тогда: $Dn = Dn_1 \otimes Dn_2 \otimes \ldots \otimes Dn_s$, а $Cn = Cn_1 \otimes Cn_2 \otimes \ldots \otimes Cn_s$.

Дискретным псевдокосинусным преобразованием вектора $X = (x_0, x_1, \ldots, x_{n-1})$ назовем вектор $Y = (y_0, y_1, \ldots, y_{n-1})$, компоненты которого определяются по формуле:

$$Y = W_n X = D_n C_n X.$$

Тогда обратное преобразованием имеет вид: $X = W_i^{-1} Y = C_n^T D_n Y$.

Вычисление двухмерного ДПКП аналогично *DCT,* то есть сначала для строк блока применяем ДПКП, затем столбцы полученной матрицы подвергаются дискретному псевдокосинусному преобразованию.

Пример. Приведите матрицу псевдокосинусного преобразования W_8.

Решение

Вспомним, что $W_8 = W_2 \otimes W_4$ и $W_8 = D_8 C_8$, где $D_8 = D_2 \otimes D_4$ $C_8 = C_2 \otimes C_4$. Учитывая определение легко получить

$$D_8 = D_2 \otimes D_4 = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{10}} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{10}} \end{pmatrix}$$

В результате получим диагональную матрицу

$$D_8 = \left(\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{5}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{5}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{5}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{5}} \right).$$

$$C_8 = C_2 \otimes C_4 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 & 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 & 1 & -2 & 2 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 2 & 1 & -1 & -2 & -2 & -1 & 1 & 2 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -2 & 2 & -1 & -1 & 2 & -2 & 1 \end{pmatrix}$$

Пример окончен.

Преобразование Уолша (WHT)

Студенты могут легко убедиться, что намного проще и быстрее считаются преобразования, основанные на импульсоподобных сигналах, которые принимают значения только ±1. Кроме того, они больше подходят для описания сигналов с нарушением непрерывности. Преобразование Уолша – это преобразование, основанное на наборе гармонических прямоугольных импульсов, которые называются функциями Уолша (рис. 8 и рис. 9).

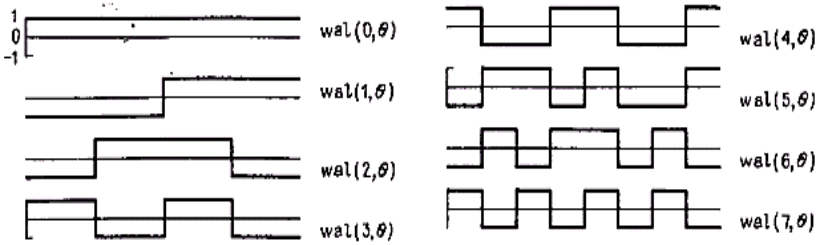


Рис. 8. Первые восемь функций Уолша с упорядочением по Уолшу *wal*(*n*, θ) (для матрицы преобразования 8×8)

Пусть $n = 2^m$ и m – натуральное число, тогда дискретным преобразованием Уолша вектора $X = (x_0, x_1, \ldots, x_{n-1})$ назовем вектор $Y = (y_0, y_1, \ldots, y_{n-1})$, компоненты которого определяются по формуле:

$$Y = \frac{1}{n} M_w X \text{ или } Y = \frac{1}{n} M_h X,$$

где M_w – матрица преобразования *Уолша с упорядочением по Уолшу* размерности $n \times n$;

M_h – матрица *преобразования Уолша –Адамара* размерности $n \times n$.

Обратное преобразование Уолша будем рассматривать, соответственно как $X = M_w Y$ или $X = M_h Y$.

Двухмерное преобразование Уолша оперирует с блоками P размером $n \times n$ и выглядит следующим образом [6]:

$$G = \frac{1}{n^2} M * P * M^T,$$

где $n = 2^m$, $m = 1, 2, 3, \ldots$;

G – матрица коэффициентов преобразования;

M – матрица преобразования, все матрицы имеют размерность $n \times n$.

Обратное преобразование Уолша описывается при помощи следующего выражения:

$$P = M * G * M^T.$$

Преобразование Уолша с упорядочением по Уолшу (WHT)$_w$

Для рассматриваемого преобразования используется следующая матрица M[10]:

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

Матрица M преобразования Уолша с упорядочением по Уолшу для $n = 4$

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{pmatrix}$$

Матрица M преобразования Уолша с упорядочением по Уолшу для $n = 8$

[10] Строки матрицы M упорядочены по количеству переходов через нуль за единицу времени.

Пример. Найти преобразование Уолша для последовательности данных $X = (1, 2, 0, 4)$.

Решение

Подставляя в явном виде матрицу M в формулу $Y = \frac{1}{n} MX$ получаем

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 0 \\ 4 \end{pmatrix} = \begin{pmatrix} 1,75 \\ -0,25 \\ 0,75 \\ -1,25 \end{pmatrix}$$

Легко убедиться, что обратное преобразование возвращает вектор X.

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \begin{pmatrix} 1,75 \\ -0,25 \\ 0,75 \\ -1,25 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 0 \\ 4 \end{pmatrix}$$

Пример окончен.

Преобразование Уолша – Адамара (WHT)$_h$ [6]

Можно сказать, что это то же преобразование Уолша, но с другим порядком функций Уолша и, следовательно, строк матрицы преобразования. Будем считать, что нам даны матрицы Адамара второго порядка ${}^2H$ и $-{}^2H$.

$${}^2H = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad -{}^2H = \begin{pmatrix} -1 & -1 \\ -1 & 1 \end{pmatrix}$$

Любую матрицу Адамара порядка $2n$ можно рекурсивно получить из ${}^2H$ как

$${}^{2n}H = \begin{pmatrix} {}^nH & {}^nH \\ {}^nH & -{}^nH \end{pmatrix}.$$

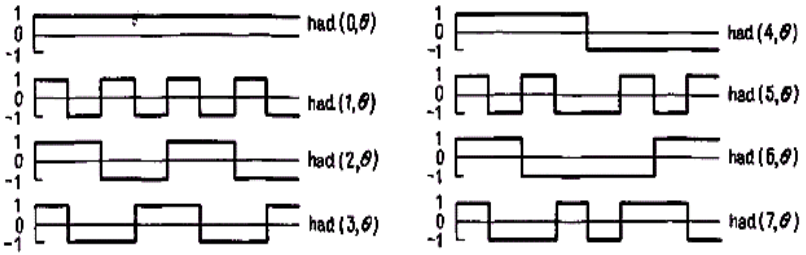


Рис. 9. Первые восемь функций Уолша с упорядочением по Адамару *had*(*n*, *θ*) (для матрицы преобразования 8×8)

$$M = {}^4H = \begin{pmatrix} {}^2H & {}^2H \\ {}^2H & -{}^2H \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

Матрица M преобразования Уолша–Адамара для $n = 4$

$$M = {}^8H = \begin{pmatrix} {}^4H & {}^4H \\ {}^4H & -{}^4H \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

Матрица M преобразования Уолша–Адамара для $n = 4$

От функций Уолша, упорядоченных по Адамару, можно перейти к функциям Уолша, упорядоченным по Уолшу [6]. Алгоритм перехода состоит из четырех шагов:

1) номер функции Уолша–Адамара представляется в двоичном коде;

2) полученные на шаге 1 векторы преобразуются – биты переставляются в обратном порядке;

3) записанные таким образом номера рассматриваются так, как если бы они были закодированы кодом Грея[11];

[11] Иногда требуются коды, у которых следующие друг за другом кодовые слова различаются только одной цифрой в одном из разрядов.

Преобразование двоичного кода в код Грея: Пусть $s = g_{n-1}g_{n-2} \ldots g_0$ – кодовое слово в n-разрядном двоичном коде Грея, а соответствующая

4) полученный на шаге 3 код Грея преобразуется в двоичный код[12]. Номерами функций Уолша упорядоченных по Уолшу являются числа полученные после преобразования двоичного вектора в число.

Пример. Постройте код Грея для чисел от 0 до 7.

Решение

Число	*Двоичный код*	*Код Грея*
0	(000)	(000)
1	(001)	(001)
2	(010)	(011)
3	(011)	(010)

Число	*Двоичный код*	*Код Грея*
4	(100)	(110)
5	(101)	(111)
6	(110)	(101)
7	(111)	(100)

Пример окончен.

Пример. Постройте преобразование из функций Уолша–Адамара в функции Уолша, упорядоченные по Уолшу для первых восьми функций Уолша–Адамара.

двоичная запись числа $s = b_{n-1}b_{n-2} \ldots b_0$. Тогда g_i может быть получена как $g_i = b_i \oplus b_{i+1}$, $0 \le i \le n-2$, $g_{n-1} = b_{n-1}$, символ $\oplus$ означает сложение по модулю 2. Например, $s = 2$, то в двоичная запись числа равна 10 и значит $b_1 = 1$; $b_0 = 0$. Следовательно, $g_1 = b_1 = 1$, $g_0 = b_0 \oplus b_1 = 1$. Таким образом, число $s = 2$ в 2-разрядном коде Грея кодируется, как 11.

[12] Преобразование кода Грея в двоичный код: Пусть $g_{n-1}g_{n-2} \ldots g_0$ – кодовое слово в n-разрядном двоичном коде Грея, соответствующее двоичному числу $b_{n-1}b_{n-2} \ldots b_0$. Начинаем с цифры самого левого разряда и двигаемся направо, принимая $b_i = g_i$, если число единиц, предшествующих g_i, четно, и $b_i = \overline{g_i}$ (черта обозначает дополнение), если число единиц, предшествующих g_i, нечетно. При этом нулевое число единиц считается четным. Например, двоичное число, соответствующее коду Грея 1001011, имеет вид 1110010 и может быть получено следующим образом:

g_6	g_5	g_4	g_3	g_2	g_1	g_0
1	0	0	1	0	1	1
↓	↓	↓	↓	↓	↓	↓
1	1	1	0	0	1	0
b_6	b_5	b_4	b_3	b_2	b_1	b_0

Решение

Номер функции Уолша–Адамара	*Номер в двоичном коде*	*Двоично-инвертированная запись. Рассматриваем ее как код Грея*	*Преобразование кода Грея в двоичный код*	*Номер функции Уолша, упорядоченной по Уолшу*
0	(000)	(000)	(000)	0
1	(001)	(100)	(111)	7
2	(010)	(010)	(011)	3
3	(011)	(110)	(100)	4
4	(100)	(001)	(001)	1
5	(101)	(101)	(110)	6
6	(110)	(011)	(010)	2
7	(111)	(111)	(101)	5

Пример окончен.

3.3. Алгоритмы сжатия изображений без потерь

3.3.1. Алгоритм LZW

При рассмотрении графических файлов студенты сталкиваются с двумя понятиями: формат файла и алгоритм сжатия. Заметим, что между форматом графического файла и алгоритмом сжатия нет однозначного соответствия. Например, модификации алгоритма LZW реализованы в огромном количестве форматов (GIF, TIFF). В то же время многие современные форматы поддерживают запись с использованием нескольких алгоритмов архивации.

Рассмотрим модификацию алгоритма LZW, которая используется в формате Gif. Описание алгоритма проведем по работе [11].

Напомним, что формат Gif разрабатывался для изображений, содержащих от 2 до 256 цветов, а файл формата Gif состоит из следующих разделов:

a) метка файла;
b) описатель экрана;
c) глобальная карта цвета;
d) описатель изображения;
e) локальная карта цвета;
f) изображение;
g) завершитель файла.

Следует отметить, что разделы d)–f) могут повторяться в файле один и более раз. Изображение хранится как последовательность значений индексов цвета. Пиксели хранятся как строки изображения слева направо. По умолчанию каждая строка изображения записана последовательно, сверху вниз (если строки хранятся в другом порядке, то это не скажется на работе алгоритма LZW).

Рассмотрим отличия алгоритма сжатия в формате Gif по сравнению со стандартным алгоритмом LZW.

> Введен код очистки, который переводит все параметры компрессии или декомпрессии и таблицы в исходное состояние. Значение этого кода равно t[13].

> Код конца информации, который явно отмечает конец потока данных изображения. В этом случае алгоритм LZW завершает свою работу. Этот код должен быть последним кодом, который выводится кодировщиком в последовательность кодов. Значение кода конца равно: *код очистки* + 1.

> Первый доступный код сжатия равен: *код очистки* + 2.

> Выводимые коды имеют переменную длину, начиная с величины *размер кода* +1 битов на код и до 12 битов на код. Таким образом, максимальное значение кода есть 4095.

Напомним, что алгоритм LZW работает с тремя объектами и при сжатии, и при разжимании: поток символов[14], поток кодов и таблица строк (словарь). Первое, что делается при сжатии, – это инициализация словаря. Дальше используется стандартный алгоритм LZW.

Поскольку применяемое в формате Gif сжатие выдает последовательность кодов переменной длины (от трех до двенадцати битов каждый), то эти коды должны быть переформатированы в последовательность восьмибитовых байтов. Это дает возможность для дополнительного сжатия изображения.

Процесс декомпрессии аналогичен стандартному алгоритму LZW.

[13] $t = 2^{\text{размер кода}}$. Размер кода – сколько битов необходимо для представления множества реальных значений пикселей, обычно это будет то же значение, что и число битов цвета. У алгоритма существуют ограничения, черно-белые изображения (с одним битом на цвет) должны считаться как имеющие размер кода, равного двум.

[14] Символ – это индекс, описывающий цвет данного пикселя.

3.4. Алгоритмы сжатия изображений с потерями

К преимуществам алгоритмов с потерями можно отнести возможность выбора необходимого коэффициента сжатия. Это часто бывает необходимо для адаптации к пропускной способности канала, емкости накопителя и т. д. Разумеется, с ростом коэффициента сжатия качество восстановленного изображения ухудшается.

В основе наиболее распространенных современных методов сжатия цифровых изображений с потерями лежат ортогональные декоррелирующие преобразования, осуществляющие разделение элементов данных на составляющие, содержащие основную информацию об изображении и определяющие малозначимые детали. Сжатие происходит за счет удаления составляющих второго типа из преобразованных элементов данных с последующим энтропийным кодированием оставшихся элементов.

Если изображение сжимается с применением блочного преобразования, то оно обычно разбивается на блоки размером 8×8 или 16×16 пикселей для ускорения вычислений, поскольку тогда каждый блок преобразуется и обрабатывается отдельно. Понятно, что здесь не учитывается корреляция между блоками изображения, хотя эта корреляция может оказаться серьезным источником информационной избыточности итогового сжатия.

3.4.1. JPEG-подобная схема компрессии

Самым распространенным в настоящее время методом сжатия изображений с потерями является JPEG, который основан на квантовании и статистическом кодировании коэффициентов дискретного косинусного преобразования (DCT). Однако данный метод сжатия может быть модифицирован.

Рассмотрим JPEG-подобную схему компрессии статического полутонового изображения. Алгоритм включает следующие этапы:

- разбивка изображения на блоки 8×8 пикселей;
- выполнение одного из рассмотренных дискретных преобразований на полученном блоке элементов изображения для декорреляции источника;

> квантование коэффициентов преобразования для снижения визуальной избыточности;

> энтропийное кодирование проквантованных значений для снижения кодовой избыточности (в качестве примера можно привести одну из модификаций кода Хаффмана).

При восстановлении, соответственно, последовательно выполняются статистическое декодирование, деквантование и обратное дискретное преобразование.

В работе [10] в качестве JPEG-подобной рассматривается следующая схема.

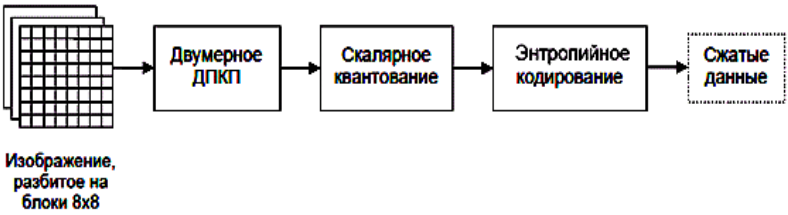


Рис. 10. JPEG-подобная схема компрессии

В работе [12] в качестве JPEG-подобной рассматривается схема, где вместо ДПКП используются преобразование Уолша, упорядоченное по Уолшу, и преобразование Уолша–Адамара.

Перед тем как мы завершим обсуждение JPEG-подобной схемы компрессии, необходимо рассмотреть, как изменится ее работа при сжатии цветного изображения. Предположим, что сжимаем 24–битовое изображение, тогда в рассмотренной схеме добавится дополнительный этап:

> цветное изображение из пространства *RGB* преобразуется в цветовое пространство *YCrCb*. Выполняем разделение изображения на три полутоновых и сжимаем их независимо друг от друга. Переход к *YCrCb* обусловлен тем фактом, что глаз чувствителен к малым изменениям яркости пикселей, но не цветности, поэтому из компоненты цветности можно удалить значительную долю информации для достижения высокого сжатия без заметного визуального ухудшения качества образа. Указанное свойство можно использовать при компрессии для увеличения скорости работы или увеличения качества компрессии (например, при сжатии

пикселей яркости использовать преобразование DCT, а при сжатии компонентов цветности использовать преобразование ДПКП или преобразования WHT).

3.4.2. Метод усеченного блочного кодирования (УБК)

Классическая реализация УБК (другое название метод BTC – Block Truncation Coding), предполагает разбиение изображения на небольшие непересекающиеся прямоугольные куски одинакового размера $m \times n$ пикселов называемые блоками [9]. Каждый такой блок обрабатывается независимо от других, поэтому рассмотрим алгоритм обработки одного блока.

Алгоритм состоит из нескольких шагов:

- пусть $k = m \times n$, а $b_1, \ldots, b_k$ – яркость соответствующего пиксела в блоке. Вычисляем среднее значение яркости пикселов C и средний квадрат E, где $C = \frac{1}{k}\sum_{i=1}^{k} b_i$ и $E = \frac{1}{k}\sum_{i=1}^{k} b_i^2$;
- для каждого блока вычисляем $\sigma^2 = E - C^2$;
- вычисляем пороговое значение яркости b_{thr}, где $b_{thr} = C$;
- определяем уровни квантования: $a^- = C - \sigma\sqrt{\frac{q}{k-q}}$ и $a^+ = C + \sigma\sqrt{\frac{k-q}{q}}$, где q – число пикселей яркость которых больше или равна пороговому значению;
- для каждого блока выполняем процесс квантования по следующему правилу:

$$s_i = \begin{cases} a^+, \text{ если } b_i \geq b_{thr} \\ a^-, \text{ если } b_i < b_{thr} \end{cases},$$

где s_i – яркость пиксела i после квантования;

- после квантования получим блок S из k элементов, элементами являются a^- или a^+. Для дальнейшего сжатия найдем блок S^* заменив в S элементы a^- на ноль, а a^+ соответственно на единицу. В сжатый файл можно отдельно записать числа a^-, a^+ и двоичный вектор $\overline{S}$ в виде числа, который строится по S^*.

Алгоритм закончен.

Таким образом, классический алгоритм BTC делит изображение на маленькие блоки пикселей, а затем сокращает количество уровней яркости в пределах каждого блока. Сокращение обычно выполняется с сохранения у декодируемого и исходного изображения каких-либо характеристик (например среднюю яркость, среднеквадратичное отклонение) или с минимизацией некоторой функции.

Студенты должны обратить внимание на то, что классическая схема УБК работает с полутоновыми изображениями, а степень сжатия и качество восстановленного изображения сильно зависят от размеров блока. Отметим, что наиболее удовлетворительные результаты были получены при применении блоков размером 4×4.

Пример. Выполнить метод усеченного блочного кодирования для изображения представленного матрицей H, где h_{ij} – яркость соответствующего пикселя.

$$H = \begin{pmatrix} 121 & 110 & 56 & 45 \\ 20 & 200 & 247 & 253 \\ 16 & 10 & 0 & 150 \\ 40 & 2 & 5 & 200 \end{pmatrix}$$

Решение

Легко заметить, что $C \approx 92{,}19$, $\sigma \approx 88{,}68$, $k = 16$, $q = 7$. Таким образом можно найти

$$a^{-} = C - \sigma\sqrt{\frac{q}{k-q}} = 92{,}19 - 88{,}68\sqrt{\frac{7}{9}} = 13{,}98$$

$$\text{и } a^{+} = C + \sigma\sqrt{\frac{k-q}{q}} = 92{,}19 + 88{,}68\sqrt{\frac{9}{7}} = 192{,}74\,.$$

После квантования получим матрицу S, битовую матрицу S^* и вектор $\overline{S}$:

$$S = \begin{pmatrix} 193 & 193 & 14 & 14 \\ 14 & 193 & 193 & 193 \\ 14 & 14 & 14 & 193 \\ 14 & 14 & 14 & 193 \end{pmatrix}, \text{ матрица } S^* = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Таким образом в выходной сжатый файл можно записать вектор $\overline{S}$ = (1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1) и два значения 193 и 14.

Пример окончен.

Рассмотренный способ определения порога и уровней квантования не является единственным. Приведем несколько модификаций классического алгоритма BTC.

– Первая модификация (AMBTC)

- пороговое значение яркости $b_{thr} = C$;
- уровни квантования $a^- = C - \frac{\alpha}{2} * \frac{k}{k-q}$ и $a^+ = C + \frac{\alpha}{2} * \frac{k}{q}$,

где b_i – яркость соответствующего пиксела в блоке, k – количество пикселов в блоке , C – среднее значение яркости пикселов, $\alpha = \frac{1}{k}\sum_{i=1}^{k}|b_i - C|$, q – число пикселей, яркость которых больше или равна пороговому значению. Процесс кодирования и декодирования аналогичен классической реализации BTC.

– Вторая модификация

- пороговое значение яркости $b_{thr} = \frac{b_{\min} + b_{\max}}{2}$;
- уровни квантования: $a^- = \frac{1}{k-q}\sum_{b_i<C} b_i$ и $a^+ = \frac{1}{q}\sum_{b_i \geq C} b_i$,

где b_i – яркость соответствующего пиксела в блоке, b_{max} и b_{min} – максимальная и минимальная яркость в блоке, C – среднее значение яркости пикселов, k – количество пикселей в блоке, q – число пикселей, яркость которых больше или равна пороговому значению. Процесс кодирования и декодирования аналогичен классической реализации BTC.

Использование УБК для сжатия изображений приводит к целому ряду специфических искажений, основным недостатком которых является то, что теряется информация об изображении, основанная на интенсивности пикселей в каждом блоке (малое число градаций яркости в пределах блока). Следовательно, изображение после УБК может иметь блочный характер.

Задачи

1. Используя статистический и динамический метод Хаффмана, закодируйте слова «дерево», «переселение», «дискуссия».

2. Используя алгоритм LZW, закодируйте слова «дискуссия», «ассамблея», «воевода», после покажите процесс декодирования.

3. Закодируйте числа {10,65,89,110} унарным кодом, кодом Левенштейна, кодом Элайеса и покажите процесс декодировки.

4. Пусть $T = 11$. Закодируйте числа {10,65,89,110} кодом Голомба.

5. Пусть $T = 16$. Закодируйте числа {10,65,89,110} кодом Голомба.

6. Предположим, что нам известны блоки размера 8×8 полутонового черно-белого изображения.

128	130	127	130	131	129	130	131
129	130	129	139	132	135	133	135
133	134	135	136	137	138	139	140
129	129	129	129	129	129	129	129
132	133	134	135	136	137	138	139
129	129	129	129	141	141	141	141
128	132	133	128	132	140	128	125
140	141	142	143	140	141	142	143

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

Выполните для них LZW кодирование.

7. Предположим, что нам известны блоки размера 8×8 полутонового 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

150	155	160	165	170	175	180	185
145	150	155	160	165	170	175	180
140	141	142	143	144	145	146	147
145	146	147	148	149	150	151	152
145	147	149	151	153	155	157	159
182	189	168	161	154	126	119	112
175	154	126	105	140	105	119	84
154	98	105	98	105	63	112	84

Выполните для них дискретное косинусное и псевдокосинусное преобразование для $n = 8$ и $n = 4$.

8. Предположим, что нам известны блоки размера 8×8 полутонового изображения.

128	130	127	130	131	129	130	131
129	130	129	139	132	135	133	135
133	134	135	136	137	138	139	140
129	129	129	129	129	129	129	129
132	133	134	135	136	137	138	139
129	129	129	129	141	141	141	141
128	132	133	128	132	140	128	125
140	141	142	143	140	141	142	143

150	155	160	165	170	175	180	185
145	150	155	160	165	170	175	180
140	141	142	143	144	145	146	147
145	146	147	148	149	150	151	152
145	147	149	151	153	155	157	159
182	189	168	161	154	126	119	112
175	154	126	105	140	105	119	84
154	98	105	98	105	63	112	84

Выполните для них преобразование Уолша, упорядоченное по Уолшу, и преобразование Уолша–Адамара для $n = 8$ и $n = 4$.

9. Закодируйте числа {15,64,89,110,125} кодом Грея.

10. Предположим, что нам известны блоки размера 8×8 полутонового изображения.

128	130	127	130	131	129	130	131
129	130	129	139	132	135	133	135
133	134	135	136	137	138	139	140
129	129	129	129	129	129	129	129
132	133	134	135	136	137	138	139
129	129	129	129	141	141	141	141
128	132	133	128	132	140	128	125
140	141	142	143	140	141	142	143

150	155	160	165	170	175	180	185
145	150	155	160	165	170	175	180
140	141	142	143	144	145	146	147
145	146	147	148	149	150	151	152
145	147	149	151	153	155	157	159
182	189	168	161	154	126	119	112
175	154	126	105	140	105	119	84
154	98	105	98	105	63	112	84

Выполните для них метод усеченного блочного кодирования (УБК) и модификации УБК.

Литература

1. Сэломон, Д. Сжатие данных, изображений и звука / Д. Сэломон. – М. : Техносфера, 2004. – 367 с.

2. Кудряшов, Б. Д. Теория информации / Б. Д. Кудряшов – СПб. : Питер, 2009. – 320 с.

3. Чернега, В. С. Сжатие информации в компьютерных сетях / В. С. Чернега. – Севастополь : СевГТУ, 1997. – 212 с.

4. Роджерс, Д. Алгоритмические основы машинной графики / Д. Роджерс. – М. : Мир, 1989. – 512 с.

5. Методы сжатия данных : Устройство архиваторов, сжатие изображений и видео / Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин. – М. : ДИАЛОГ-МИФИ, 2002. – 384 с.

6. Ахмед, Н. Ортогональные преобразования при обработке цифровых сигналов / Н. Ахмед, К. Р. Рао. – М. : Связь, 1980. – 248 с.

7. Айфичер, Э., Цифровая обработка сигналов : Практический подход / Э. Айфичер, Б. Джервис. – М. : Вильямс, 2004. – 992 с.

8. Умняшкин, С. В. О модификации дискретного косинусного преобразования / С. В. Умняшкин // Известия Тул. гос. ун-та. Серия : Математика. Механика. Информатика. Вып. 1. – Тула : ТулГУ, 1998. – Т. 4. – С. 143–147.

9. Salomon, D. Data Compression, Springer-Verlag / D. Salomon. – L. : London Limited, 2007. – 1093 с.

10. Умняшкин, С. В., Алгоритм сжатия изображений на основе дискретного псевдокосинусного преобразования / С. В. Умняшкин, В. В. Курина // Цифровая обработка сигналов. – 2009. – № 3. – С. 2–7.

11. Романов, В. Ю. Популярные форматы файлов для хранения графических изображений на IBM PC / В. Ю. Романов. – М. : Унисех, 1992. – 156 с.

12. Карагодин, М. А. Алгоритм сжатия изображений на основе функций Уолша / М. А. Карагодин, А. Н. Осокин // Информационные системы и технологии : материалы межд. конф. ИСТ'2003. – Новосибирск, 2003. – Т. 2. – С. 126–130.

Оглавление

Учебное издание

Методы сжатия информации: текст и изображение

Методические указания

Составитель
Краснов Михаил Владимирович

Редактор, корректор М. Э. Левакова
Верстка Е. Б. Половковой

Подписано в печать 09.10.14. Формат 60×84 1/16.
Усл. печ. л. 2,56. Уч.-изд. л. 2,0.
Тираж 20 экз. Заказ

Оригинал-макет подготовлен
в редакционно-издательском отделе ЯрГУ.

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