Словарные методы кодирования

Словарные методы

- Статистические методы компрессии используют статистическую модель данных, и качество сжатия информации напрямую зависит от того, насколько хороша была эта модель.

- Методы основанные на словарном подходе, не рассматривают статистические модели.

- Вместо этого они выбирают некоторые последовательности символов, сохраняют их в словаре, а все последовательности кодируются в виде меток (кодов, чисел), используя словарь.

Алгоритм RLE.

- *Первый вариант алгоритма*
- Данный алгоритм необычайно прост в реализации. Групповое кодирование — от английского **Run Length Encoding (RLE)** — один из самых старых и самых простых алгоритмов архивации графики.

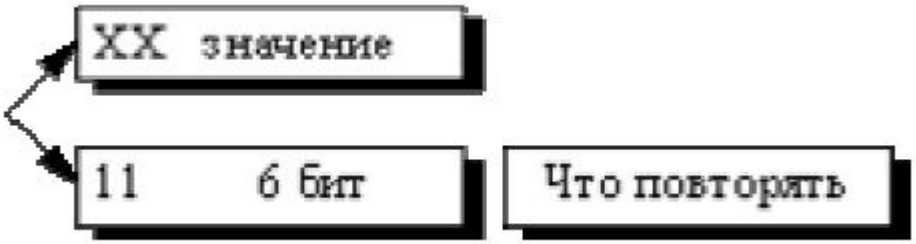
- Изображение в нем вытягивается в цепочку байт по строкам растра.

- Само сжатие в RLE происходит **за счет того, что в исходном изображении встречаются цепочки одинаковых байт.**

- Замена их на пары **<счетчик повторений, значение>** уменьшает избыточность данных.

Первый вариант алгоритма RLE

- В данном алгоритме признаком счетчика служат единицы в двух верхних битах
- считанного файла:



- Соответственно оставшиеся 6 бит расходуются на счетчик, который может принимать значения от 1 до 64. Строку из 64 повторяющихся байтов мы превращаем в два байта, т.е. сожмем в 32 раза.

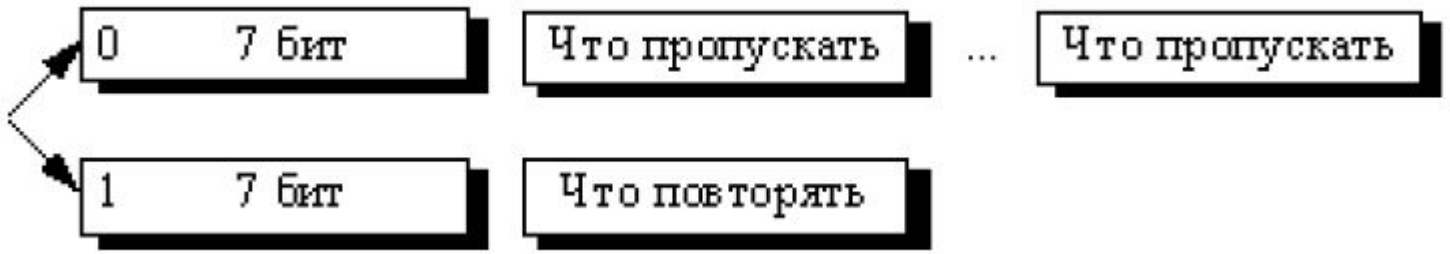
Первый вариант RLE

- Алгоритм рассчитан на деловую графику — изображения с большими областями повторяющегося цвета.
- Ситуация, когда файл увеличивается, для этого простого алгоритма не так уж редка.
- Ее можно легко получить, применяя групповое кодирование к обработанным цветным фотографиям.
- Для того, чтобы увеличить изображение в два раза, его надо применить к изображению, в котором значения всех пикселов больше двоичного 11000000 и подряд попарно не повторяются.
- Данный алгоритм реализован в формате PCX.

RLE

Второй вариант алгоритма

- Второй вариант этого алгоритма имеет больший максимальный коэффициент архивации и меньше увеличивает в размерах исходный файл.
- Признаком повтора в данном алгоритме является единица в старшем разряде соответствующего байта:



RLE 2

- Как можно легко подсчитать, в лучшем случае этот алгоритм сжимает файл в 64 раза (а не в 32 раза, как в предыдущем варианте), в худшем увеличивает на 1/128. Средние показатели степени компрессии данного алгоритма находятся на уровне показателей первого варианта.

LZW

- Название алгоритм получил по первым буквам фамилий его разработчиков —
- Lempel, Ziv и Welch.
-
- Сжатие в нем, в отличие от RLE, осуществляется уже за счет одинаковых цепочек байт.

Алгоритм LZW

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

- LZW реализован в форматах GIF и TIFF.
- LZW - это способ сжатия данных, который извлекает преимущества при повторяющихся цепочках данных.
- Поскольку растровые данные обычно содержат довольно много таких повторений, LZW является хорошим методом для их сжатия и раскрытия

LZW-сжатие

- **1. Инициализация цепочки символов.**
- Выбираем размер кода (количество бит) и определяем сколько возможных значений могут принимать наши символы.
- Положим код размера равным 12 битам, что означает возможность запоминания 0FFF, или 4096, элементов в нашей таблице цепочек.
- Предположим , что мы имеем 32 возможных различных символа. (Это соответствует, например, картинке с 32 возможными цветами для каждого пиксела.)
- Чтобы инициализировать таблицу, мы установим соответствие кода #0 символу #0, кода #1 символу #1, и т.д., до кода #31 и символа #31.
- На самом деле мы указали, что каждый код от 0 до 31 является корневым. Больше в таблице не будет других кодов, обладающих этим свойством.

2. Процесс сжатия

- Мы считываем последовательно символы входного потока и проверяем, есть ли в созданной нами таблице строк такая строка.
- Если строка есть, то мы считываем следующий символ, а если строки нет, то мы заносим в поток код для предыдущей найденной строки, заносим строку в таблицу и начинаем поиск снова.

Пример

- Пусть мы сжимаем последовательность
- 45, 55, 55, 151, 55, 55, 55.
- Тогда, поместим в выходной поток сначала код очистки <256>, потом добавим к изначально пустой строке “45” и проверим, есть ли строка “45” в таблице.
- Поскольку мы при инициализации занесли в таблицу все строки из одного символа, то строка “45” есть в таблице.

Пример: процесс сжатия

- Далее мы читаем следующий символ 55 из входного потока и проверяем, есть ли строка “45, 55” в таблице.
- Такой строки в таблице пока нет. Мы заносим в таблицу строку “45, 55” (с первым свободным кодом 258) и записываем в поток код <45>.

Формирование таблицы

- Можно коротко представить архивацию так:
- “45” — есть в таблице;
- “45, 55” — нет. Добавляем в таблицу <258>“45, 55”. В поток: <45>;
- “55, 55” — нет. В таблицу: <259>“55, 55”. В поток: <55>;
- “55, 151” — нет. В таблицу: <260>“55, 151”. В поток: <55>;
- “151, 55” — нет. В таблицу: <261>“151, 55”. В поток: <151>;
- “55, 55” — есть в таблице;
- “55, 55, 55” — нет. В таблицу: “55, 55, 55” <262>. В поток: <259>;
- Последовательность кодов для данного примера, попадающих в выходной поток:
- <256>, <45>, <55>, <55>, <151>, <259>.

3. Декомпрессия

- Особенность LZW заключается в том, что для декомпрессии нам не надо сохранять таблицу строк в файл для распаковки.

- Алгоритм построен таким образом, что мы в состоянии восстановить таблицу строк, пользуясь только потоком кодов.

3. Декомпрессия

- Мы знаем, что для каждого кода надо добавлять в таблицу строку, состоящую из уже присутствующей там строки и символа, с которого начинается следующая строка в потоке.

Ziv-Lempel Coding (ZL or LZ)

- Авторы: J. Ziv и A. Lempel (1977).

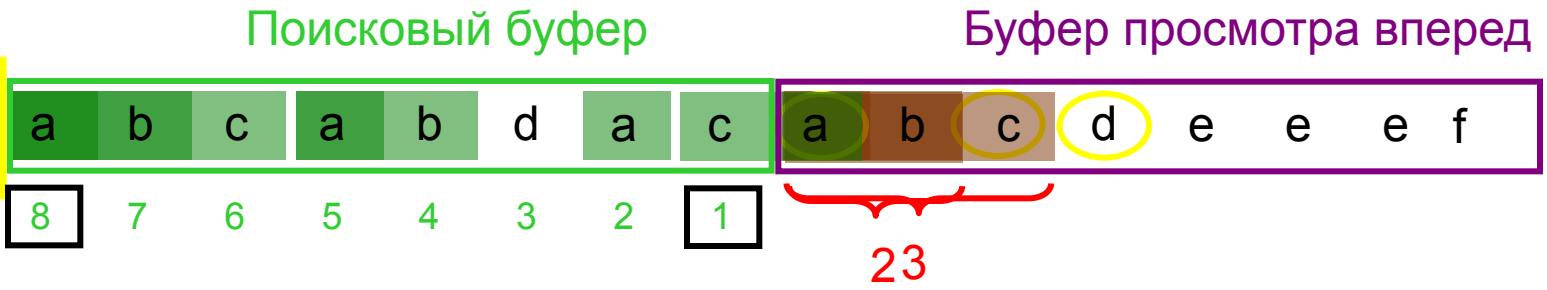
- Техника адаптивного словаря.
  - Накопление предыдущих символов в буфер.
  - Поиск текущей последовательности символов в коде .
  - Если найдена, то передается величина сдвига буфера и длина

- Основная идея LZ77 состоит в том, что второе и последующие вхождения некоторой строки символов в сообщении заменяются ссылками на ее первое вхождение.
- LZ77 использует уже просмотренную часть сообщения как словарь. Чтобы добиться сжатия, он пытается заменить очередной фрагмент сообщения на указатель в содержимое словаря.

LZ77

- LZ77 использует "скользящее" по сообщению окно, разделенное на две неравные части. Первая, большая по размеру, включает уже просмотренную часть сообщения. Вторая, намного меньшая, является буфером, содержащим еще незакодированные символы входного потока

LZ 77



Тройка выхода <offset, length, next>

Передает в канала: 8 3 d 0 0 e 1 2 f

Если размер буфера N и размер алфавита = M
нам требуется

$$\lceil \log(N+1) \rceil + \lceil \log(N+1) \rceil + \lceil \log M \rceil$$

Бит кода в тройке выхода.

PKZip, Zip, Lharc,
PNG, gzip, ARJ

LZ 77

```
clc;
clear all;
close all;
str='cabracadabrarrarrad';
strl=length(str);
code={'a','000';'b','001';'c','010';'d','011';'r','100'};
sbl=7;
labl=6;
i=1;
pos=[];
while i+sbl<strl
sb=str(i:i+sbl-1);
if i+sbl+labl-1<strl
lab=str(i+sbl:i+sbl+labl-1);
else
lab=str(i+sbl:end);
end
sp=lab(1);
pos=[];
for j=1:sbl
if sp~=sb(j)
offset=0;
ml=0;
%send code
else
pos=[pos j];
end
end
Ml=[];
for k=1:length(pos)
count=0;
m=1;
flag=0;
for l=pos(k):sbl+labl
while m<labl
if lab(m)==str(i+l-1)
count=count+1;
break;
else
flag=1;
break;
end
end
m=m+1;
if flag==1
break;
end
end
Ml=[Ml count];
end
if length(pos)==0
c=str(i+ml+sbl);
Source : www.vitedu.in
Source : www.vitedu.in
i=i+1;
else
ml=max(Ml);
c=str(i+ml+sbl);
i=i+ml+1;
end
if length(Ml)~=0
n=2;
for m=1:length(Ml)-1
if Ml(n)<Ml(m)
n=m;
else
n=n;
end
end
offset=sbl-pos(n)+1;
end
for m=1:5
if strcmp(c,code(m,1))==1
p=m;
end
end
output={offset ml code{p,2}};
disp(output);
end
```

К недостаткам алгоритма LZ77 следует отнести следующие:

1) невозможность кодирования подстрок, находящихся друг от друга на расстоянии, большем длины словаря;

2) длина подстроки, которую можно закодировать, ограничена размером буфера.

Эффективность

$$R(f_{77}, X) = O\left(\frac{\log\log W}{\log W}\right) \text{ при } W \to \infty$$

(здесь X – приизвольный марковский источник и W – длина скользящего окна).

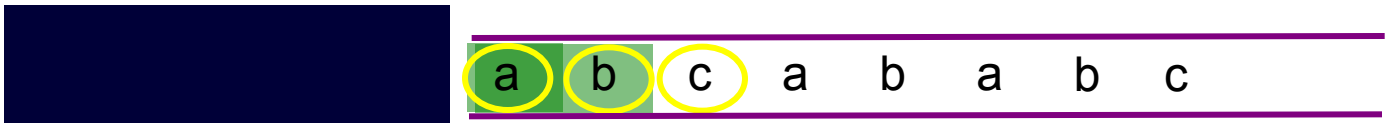
LZ 78

- **LZ 78 не использует "скользящее" окно, он хранит словарь из уже просмотренных фраз.**
- При старте алгоритма этот словарь содержит только одну пустую строку (строку длины нуль).
- Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока входит целиком в одну из фраз словаря.
- Как только эта строка перестанет соответствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введенного символа содержала входную строку, и символа, нарушившего совпадение

LZ 78

- Затем в словарь добавляется введенная подстрока. Если словарь уже заполнен, то из него предварительно удаляют менее всех используемую в сравнениях фразу.
- Ключевым для размера получаемых кодов является размер словаря во фразах, потому что каждый код при кодировании по методу LZ78 содержит номер фразы в словаре. Из последнего следует, что эти коды имеют постоянную длину, равную округленному в большую сторону двоичному логарифму размера словаря +8 (это количество бит в байт-коде расширенного ASCII).

LZ 78



Код выхода двойка <dictionary index, next>

Передача в канал: 0 a 0 b 0 c 1 b 4 c

Декодирование: a b c a b a b c

Словарь:

1	a
2	b
3	c
4	a b
5	a b c

Требуется не ограничивать размер словаря

Ziv-Lempel-Welch (LZW)-Codes

Идея: вместо последовательностей букв передаются номера слов в некотором словаре.

Кодер и декодер в процессе работы синхронно формируют этот словарь.

На каждом шаге словарь пополняется новым словом, которое до этого в словаре отсутствовало, но является продолжением на одну букву одного из слов словаря.

Ziv-Lempel-Welch (LZW)-Codes

- Пусть X={0,1,…,M-1} – алфавит источника и на выходе источника наблюдается последовательность $x_1, x_2, \ldots$.

- Будем считать, что в начале работы кодера каждая из букв алфавита является словом длины 1(один) и входит в состав словаря.

- На каждом следующем шаге находим самое длинное слово, совпадающее с началом подлежащей кодированию последовательности.

Ziv-Lempel-Welch (LZW)-Codes

- Пусть *l* – длина совпадения.
- Эти *l* букв передаются в виде ссылки на соответствующее слово словаря. Если объем словаря равен V, то для передачи этой ссылки достаточно $\lceil \log(V-1) \rceil$ бит.
- Словарь пополняется новым словом, которое получается дописыванием к использованному на данном шаге слову следующей за ним в потоке кодируемых данных буквы.
-

LZW

Input sequence:	a	b	c	a	b	a	b	c

Выход: dictionary index (индекс словаря)

Передача:

1	2	3	5	5

Декодирование:

a	b	c	a b	a b

Словарь кодера:

1	a	6	bc
2	b	7	ca
3	c	8	aba
4	d	9	abc
5	a b		

Словарь декодера:

1	a	6	bc
2	b	7	ca
3	c	8	aba
4	d		
5	a b		

Алгоритм LZW

Задан алфавит источника $X = \{0, 1, ..., M-1\}$;
Input: Длина последовательности n;
Последовательность источника $\boldsymbol{x} = \boldsymbol{x}_1^n$;
Output: Кодовое слово $\boldsymbol{c}$ для $\boldsymbol{x}$;

`Инициализация:`
$N = 0$;
$\boldsymbol{c}$ – "пустое" слово;
Словарь состоит из M слов длины 1, т.е. из букв алфавита X
Число слов в словаре $c = M$.

while $N < n$ **do**

- Находим максимальное l такое, что $\boldsymbol{x}_{N+1}^{N+l}$ совпадает с j-м словом словаря при для некотором $j < c$ (на первом шаге допускается $j = c$).
- К кодовому слову дописывается номер словарного слова j в виде двоичной последовательности длины $\lceil \log(c-1) \rceil$ (на первом шаге дописывается последовательность длины $\lceil \log c \rceil = \lceil \log M \rceil$).
- В словарь дописывается новое слово длины $l+1$ вида $\boldsymbol{x}_{N+1}^{N+l+1} = (\boldsymbol{x}_{N+1}^{N+l}, x_{N+l+1})$.
- $N \leftarrow N + l$;
- $c \leftarrow c + 1$;

end

LZW

```
im = rgb2gray(imread('lenna.png'));
[h w] = size(im);
Q = 16;
X = im(:);
len = numel(X);

lenh = len_huffman(X);
lena = len_arith(X);
fprintf('Naively:\n');
fprintf('Original: %d,\n Huffman: %d (Ratio: %f),\n Arithmetic: %d (Ratio: %f)\n', ...
    len*8, lenh, lenh/(len*8), lena, lena/(len*8));

% Do simple differencing transform
X0 = double(im(1:2:end, 1:2:end));
X1 = double(im(2:2:end, 1:2:end)) - X0;
X2 = double(im(1:2:end, 2:2:end)) - X0;
X3 = double(im(2:2:end, 2:2:end)) - X0;

% Quantise
X1 = round(X1 ./ Q);
X2 = round(X2 ./ Q);
X3 = round(X3 ./ Q);
```

LZW

```
X = X0(:);
D = [X1(:); X2(:); X3(:)];

lenhX = len_huffman(X);
lenhD = len_huffman(D);
lenh = lenhX + lenhD;

lenaX = len_arith(X);
lenaD = len_arith(D);
lena = lenaX + lenaD;

fprintf('\n\nAfter transform:\n');
fprintf('Original: %d,\n Huffman: %d (Ratio: %f),\n Arithmetic: %d (Ratio: %f)\n', ...
    len*8, lenh, lenh/(len*8), lena, lena/(len*8));

% Reconstruct image
imrec = zeros(size(im));
imrec(1:2:end, 1:2:end) = X0;
imrec(2:2:end, 1:2:end) = X0 + X1*Q;
imrec(1:2:end, 2:2:end) = X0 + X2*Q;
imrec(2:2:end, 2:2:end) = X0 + X3*Q;

im = double(im);
out = [im imrec];
diff = abs(im - imrec);
figure(1); clf; sc(out);
figure(2); clf; sc(diff);
```

figure(1); clf; sc(out);

```
diff = abs(im - imrec);
figure(2); clf; sc(diff);
```