Пример использования метода Лемпеля-Зива

Основная идея

Входная (сжимаемая) последовательность символов - последовательность строк, содержащих произвольное количество символов.

Идея (словарных методов) - замена строк символов на такие коды, что их можно трактовать как индексы строк некоторого словаря.

Образующие словарь строки - фразы.

При декодировании (распаковке) - обратная замена: индекса на соответствующую фразу словаря.

Словарь – набор фраз, которые будут встречаться в обрабатываемой последовательности.

Индексы фраз должны быть построены таким образом, чтобы их представление занимало меньше места, чем требуют замещаемые строки. За счет этого и происходит сжатие.

Алгоритмы словарного сжатия Лемпеля-Зива появились во второй половине 70-х гг. Это были так называемые алгоритмы LZ77 и LZ78, разработанные совместно Зивом (Ziv) и Лемпелом (Lempel).

LZ77 и LZ78 являются универсальными алгоритмами :

словарь формируется на основании уже обработанной части входного потока, т. е. адаптивно.

Методы LZ являются самыми популярными среди всех методов сжатия данных: практически все реально используемые словарные алгоритмы относятся к семейству Лемпела - Зива.

Алгоритм LZ77 является родоначальником словарных схем - алгоритмов **со скользящим словарем** (словарь "скользит" по входному потоку данных), или **скользящим окном**: в качестве словаря используется блок уже закодированной последовательности.

Скользящее окно имеет длину **n**, т. е. в него помещается **n** символов, и состоит из двух частей:

- последовательности длины $n_1=n-n_2$ уже закодированных символов (словарь);
- упреждающего буфера (буфера предварительного просмотра, *lookahead*), длины n_2 – **буфер кодирования**

Формальное представление алгоритма.

Пусть к текущему моменту времени закодировано **t** символов $S_1, S_2, \ldots, S_t$. Тогда словарем будут являться n_1 предшествующих символов: $S_{t-(n1-1)}, S_{t-(n1-1)+1}, \ldots, S_t$.

В буфере находятся ожидающие кодирования символы

$S_{t+1}, S_{t+2}, \ldots, S_{t+n2}$. Если $n_2 >= t$, то словарем будет являться вся уже обработанная часть входной последовательности.

Нужно найти самое длинное совпадение между строкой буфера кодирования, начинающейся с символа S_{t+1}, и всеми фразами словаря.

- Эти фразы могут начинаться с любого символа $S_{t-(n1-1)}$, $S_{t-(n1-1)+1}$, …, S_t , выходить за пределы словаря, вторгаясь в область буфера.
- Буфер не может сравниваться сам с собой.
- Длина совпадения не должна превышать размера буфера. Полученная в результате поиска фраза $S_{t-(p-1)}$, $S_{t-(p-1)+1}$, $S_{t-(p-1)+(q-1)}$ кодируется с помощью двух чисел:

1) смещения (*offset*) от начала словаря, **p**;

2) длины соответствия, или совпадения (*match length*),**q**

p и **q** - указатели (ссылки), однозначно определяют фразу.

Дополнительно в выходной поток записывается символ **s**, следующий за совпавшей строкой буфера.

длина кодовой комбинации (триады – p,q,s) на каждом шаге:

$$l(C_i) = \log_N n_1 + \log_N n_2 + 1$$

N – мощность алфавита

- После каждого шага окно смещается на **q+1**символов вправо и осуществляется переход к новому циклу кодирования.
- Величина сдвига объясняется тем, что мы реально закодировали именно **q+1** символов: **q** с помощью указателя и 1 - с помощью тривиального копирования - кодирование лексических единиц группами (байт) фиксированной длины .
- Передача одного символа в явном виде (**s**) позволяет разрешить проблему обработки еще ни разу не встречавшихся символов, но существенно увеличивает размер сжатого блока.

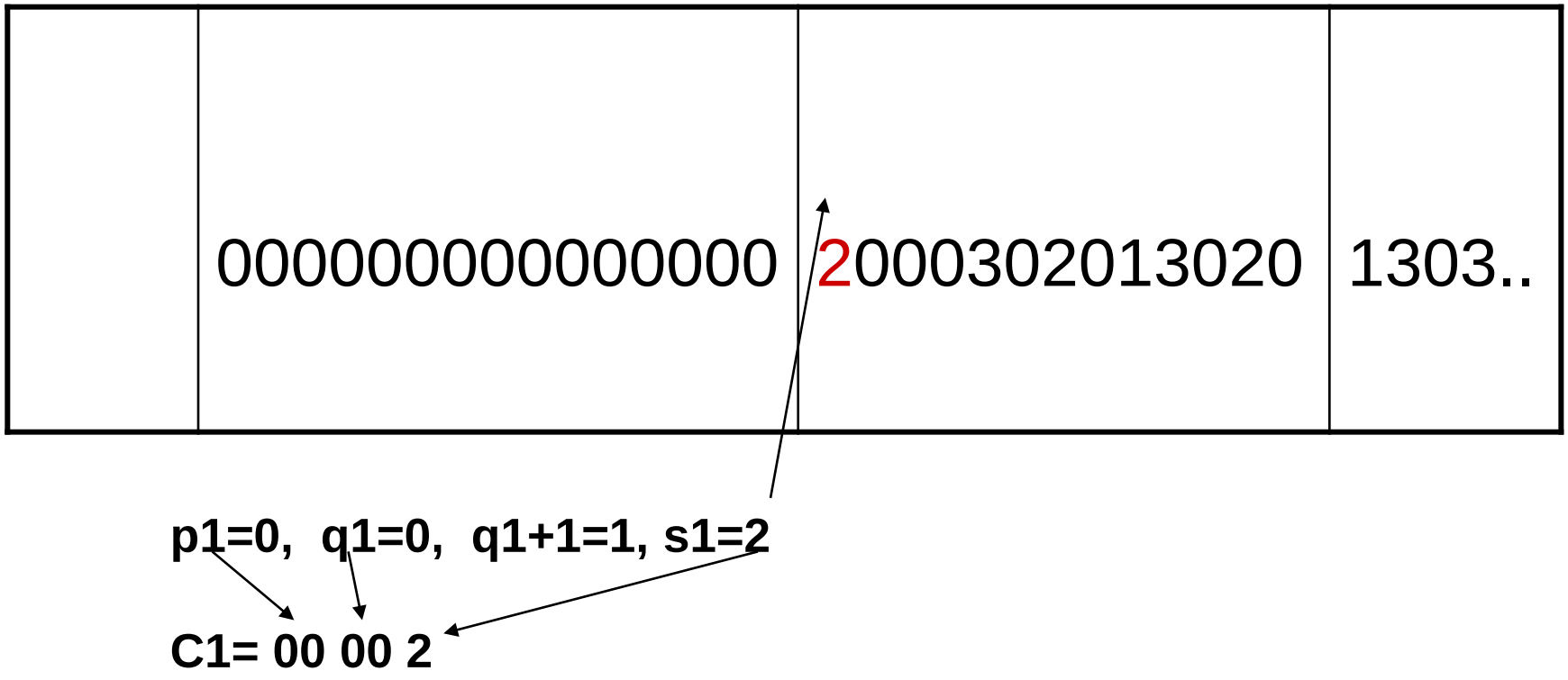
Пример.

- Используется алфавит **A**= {0,1,2,3}; N=4
- длина словаря **n1**=15
- длина буфера данных (кодирования) **n2**=13
- для обозначения **p** и **q** используется четверичная система счисления
- длина кодовой комбинации на каждом шаге $l(C_i) = \log_N n1 + \log_N n2 + 1 = 2+2+1 = 5$

Входная последовательность:

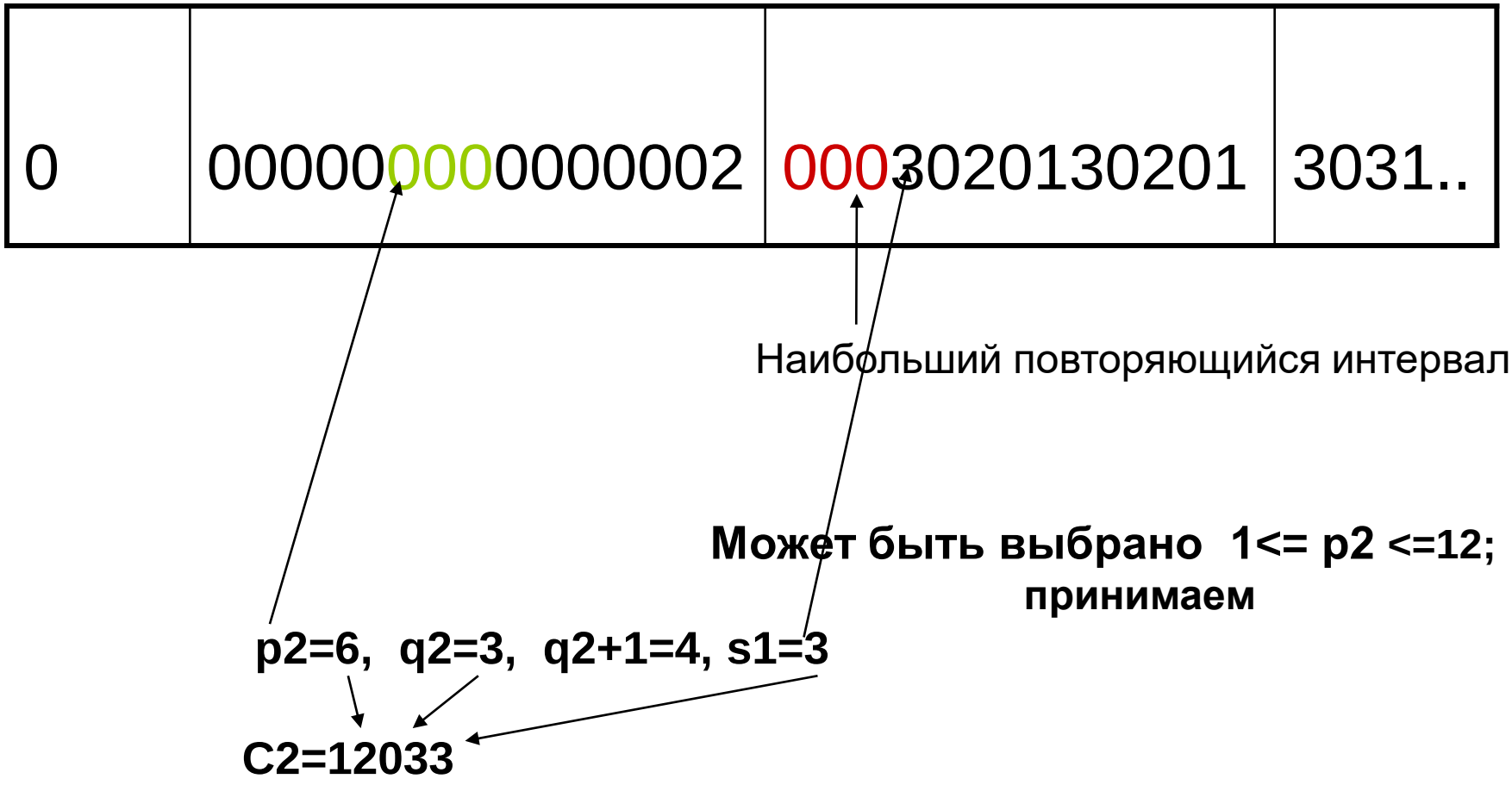
- S=2000302013020130313031303130313333333

Сжатие. Шаг 1


000000000000000
2000302013020
1303..
p1=0, q1=0, q1+1=1, s1=2
C1= 00 00 2

Шаг 2

Сдвиг буфера на 1 разряд



Шаг 3

Сдвиг буфера на 4 позиции

00000	000000000020003	02013020013031	30313. .

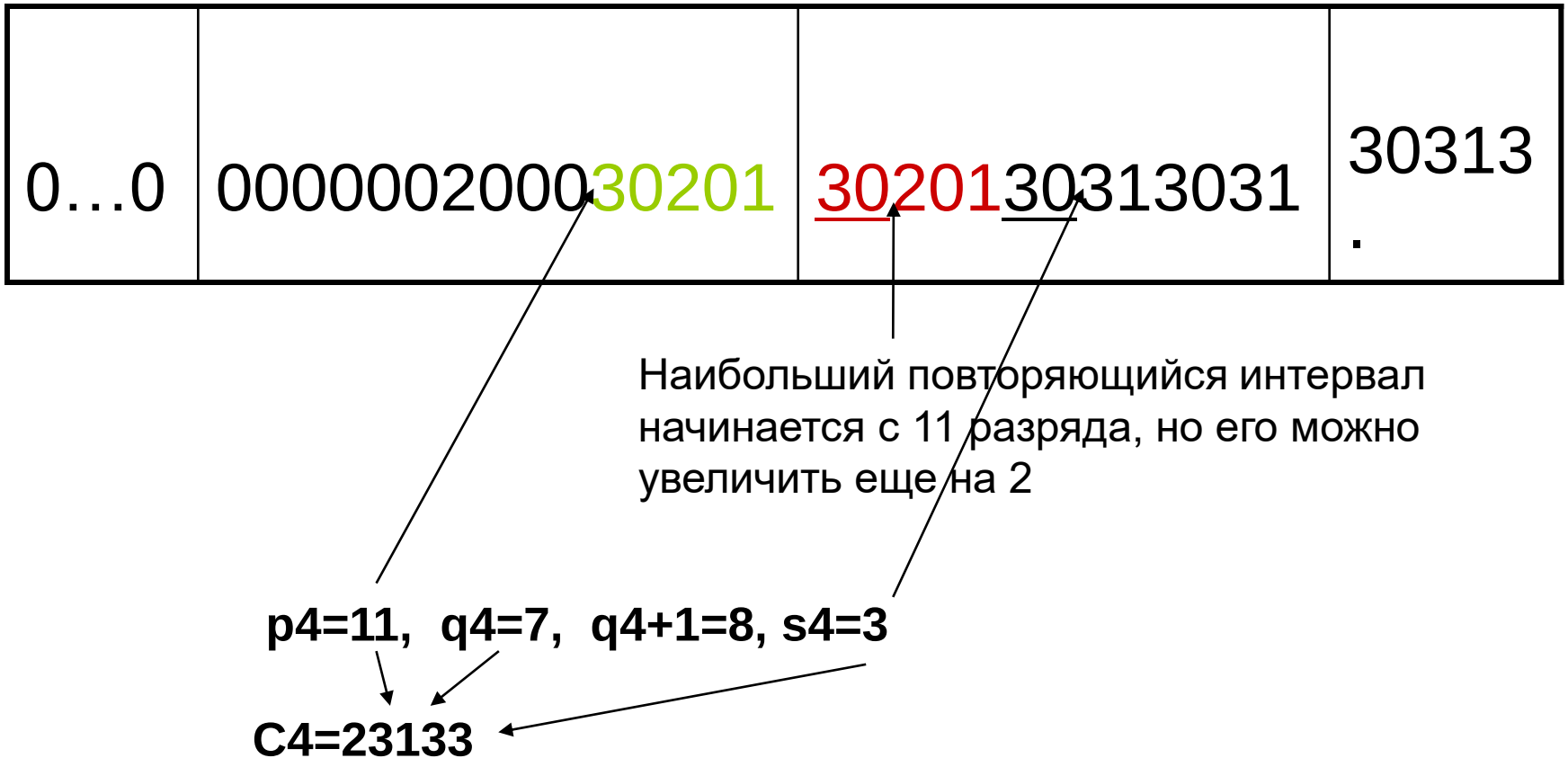
Наибольший повторяющийся интервал

p3=10, q3=3, q3+1=4, s1=1

C3=22031

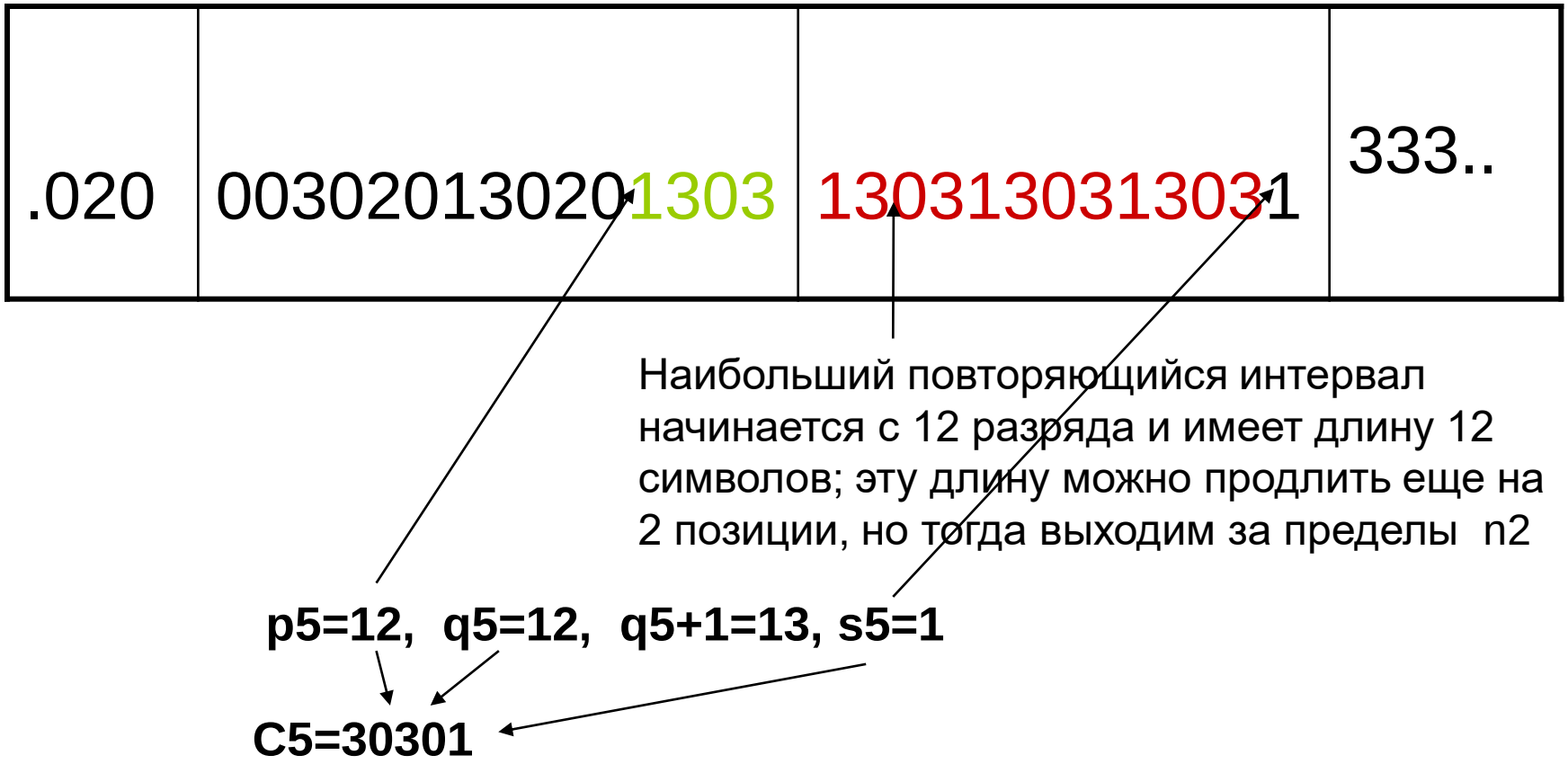
Шаг 4

Сдвиг буфера на 4 позиции



Шаг 5

Сдвиг буфера на 8 позиций

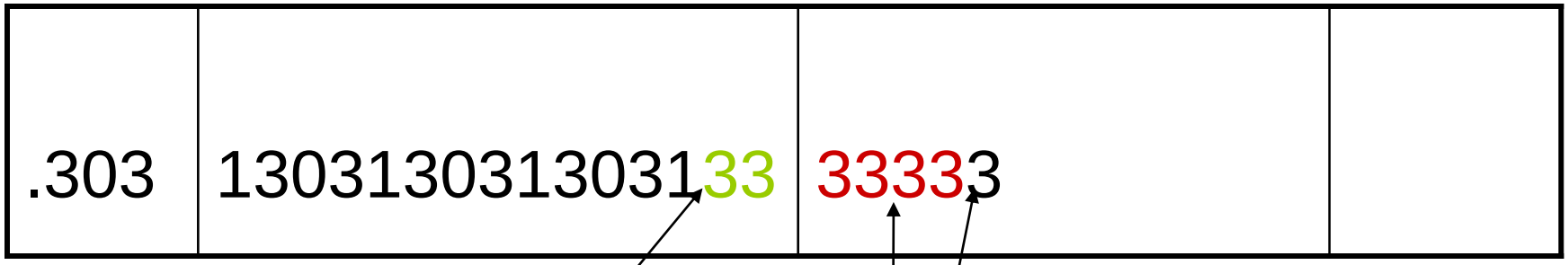


Шаг 6

Сдвиг буфера на 13 позиций

Шаг 7

Сдвиг буфера на 2 позиции



Наибольший повторяющийся интервал начинается с 14 разряда и его длина – 4 символа

p7=14, q7=4, q7+1=5, s7=3

C7=32103

На выходе; C=0000212033220312313230301020133 2103

Декомпрессия

- Используется один буфер (в примере длиной 15 разрядов)
- Изначально буфер заполнен нулями
- На входе декомпрессора: **C=00002 12033 22031 23132 30301 02013 32103**

Шаг 1

0000000000000000

Анализируется первая триада:**00002**
В младший разряд буфера записывается символ **2**

0000000000000002

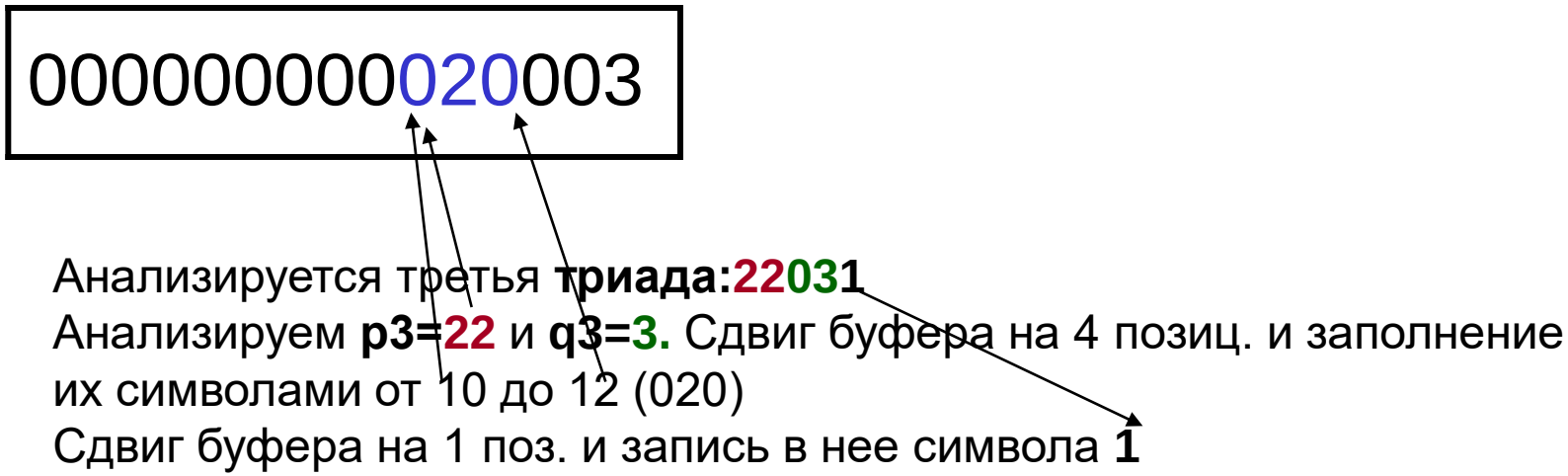
Шаг 2

000000000000002

Анализируется вторая триада:**12033**
Сдвиг буфера на **3** символа, заполнение его тремя символами предыдущего состояния буфера, начиная с поз. p1=**6** (0)
Сдвиг буфера на 1 поз. и заполнение ее символом **3**

000000000020003

Шаг 3


000000000020003
Анализируется третья **триада:22031**
Анализируем **p3=22** и **q3=3.** Сдвиг буфера на 4 позиц. и заполнение их символами от 10 до 12 (020)
Сдвиг буфера на 1 поз. и запись в нее символа **1**
000000200030201

И т.д.

Эффективность метода для примера

- Объем до сжатия – 37 символов,
- Объем после сжатия – 35 символов

Шаг 1

0000000000000000

Анализируется первая триада:**00002**
В младший разряд буфера записывается символ **2**

0000000000000002

Шаг 1

000000000000000

Анализируется первая триада:**00002**

В младший разряд буфера записывается символ **2**

000000000000002