



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное учреждение высшего образования  
**«Дальневосточный федеральный университет»**  
(ДВФУ)

---

## **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Департамент математического и компьютерного моделирования**

### **О Т Ч Е Т**

по практическому заданию по дисциплине  
«Алгоритмы и структуры данных»  
об исследовании алгоритма Лемпеля-Зива-Велча

направление подготовки 09.03.03 «Прикладная информатика»  
профиль «Прикладная информатика в компьютерном дизайне»

Отчет защищен

с оценкой: \_\_\_\_\_

Регистрационный номер \_\_\_\_\_

« \_\_\_\_ » \_\_\_\_\_ 2022г.

Выполнил студент  
гр. № Б9121-09.03.03пикд

\_\_\_\_\_ Кирилов Д. Ю.  
(подпись)

Руководитель

\_\_\_\_\_  
(должность, ученое звание)

\_\_\_\_\_ (подпись) \_\_\_\_\_ (ФИО)

« \_\_\_\_ » \_\_\_\_\_ 2022г.

г. Владивосток  
2022 г.

## Оглавление

1. Введение .....	3
1.1. Преимущества алгоритма LZW .....	3
1.2. Недостатки алгоритма LZW .....	3
2. Применение .....	4
3. Сжатие.....	5
3.1. Пример .....	5
3.2. Псевдокод алгоритма сжатия .....	6
4. Распаковка .....	8
4.1. Пример .....	8
4.2. Возможные проблемы при декодировании .....	9
4.3. Псевдокод алгоритма распаковки .....	10
5. Оценка алгоритма.....	11
6. Тесты.....	12
Тест 1 .....	12
Тест 2 .....	12
Тест 3 .....	12
Тест 4 .....	12
Тест 5 .....	12
Тест 6 .....	12
Тест 7 .....	12
Тест 8 .....	12
Тест 9 .....	12
Тест 10–13.....	12
Тест 14–17.....	13
Список литературы .....	14

# 1. Введение

Алгоритм **Лемпеля—Зива—Велча (Lempel-Ziv-Welch, LZW)** — это универсальный алгоритм сжатия данных без потерь, созданный Авраамом Лемпелем (англ. Abraham Lempel), Яковом Зивом (англ. Jacob Ziv) и Терри Велчем (англ. Terry Welch). Оригинальная версия метода была создана Лемпелем и Зивом в 1978 году (LZ78) и доработана Уэлчем в 1984 году.

Если взглянуть почти на любой файл данных в компьютере, просматривая символ за символом, то можно заметить множество повторяющихся элементов. LZW — это метод сжатия данных, который воспользовался этим повторением.

Как и в любом адаптивном/динамическом методе сжатия, идея заключается в том, чтобы (1) начать с исходной модели, (2) читать данные по частям, (3) обновлять модель и кодировать данные по мере продвижения. LZW — алгоритм сжатия на основе "словаря", т. е. LZW кодирует данные, обращаясь к словарю. Таким образом, чтобы закодировать подстроку, в выходной файл нужно записать только одно кодовое число, соответствующее индексу этой подстроки в словаре.

## 1.1. Преимущества алгоритма LZW

- Алгоритм является однократным.
- Для декомпрессии не надо сохранять таблицу строк в файл для распаковки. Алгоритм построен таким образом, что мы в состоянии восстановить таблицу строк, пользуясь только потоком кодов.
- Высокая степень сжатия.
- Достаточно высокая скорость сжатия и распаковки.

## 1.2. Недостатки алгоритма LZW

- Алгоритм не проводит анализ входных данных.

## 2. Применение

На момент своего появления алгоритм LZW давал лучший коэффициент сжатия для большинства приложений, чем любой другой хорошо известный метод того времени. Он стал первым широко используемым на компьютерах методом сжатия данных.

Алгоритм (а точнее его модификация, LZC, см. ниже) был реализован в программе compress, которая стала более или менее стандартной утилитой Unix-систем приблизительно в 1986 году. Несколько других популярных утилит-архиваторов также используют этот метод или близкие к нему.

В 1987 году алгоритм стал частью стандарта на формат изображений GIF. Он также может (опционально) использоваться в формате TIFF. Помимо этого, алгоритм применяется в протоколе модемной связи v.42bis и стандарте PDF (хотя по умолчанию большая часть текстовых и визуальных данных в PDF сжимается с помощью алгоритма Deflate).

### 3. Сжатие

LZW-алгоритм начинает работу с исходного словаря символов и использует его в качестве "стандартного" набора. В начале словарь представляет собой односимвольные цепочки, которые могут встретиться в процессе работы алгоритма. Процесс кодирования состоит в том, что алгоритм считывает из входного потока символ и проверяет, есть ли этот символ в словаре. Встретив символ в словаре, алгоритм кодирует их в виде числа, которое представляет собой индекс в этом самом словаре. Всякий раз, встречая новую подстроку (например, "ab"), он добавляет ее в словарь; каждый раз, когда ему попадаетея подстрока, которая ранее уже встречалась, он просто считывает новый символ и выполняет его конкатенацию с текущей строкой, чтобы получить новую подстроку.

Говоря простым языком, LZW-алгоритм считывает очередной непрочитанный символ и пытается найти его в словаре. Если он в словаре есть, то алгоритм пытается его удлинить, путём добавления следующего, и ищет в словаре уже удлиненную цепочку. Как только алгоритм находит цепочку, которая в коротком варианте уже записана в словаре, а в удлинённом нет, короткий вариант кодируется индексом, который уже есть в словаре, а удлиненный вариант заносится в словарь.

#### 3.1. Пример

Предположим, что наш поток, который мы хотим сжать, это «abdabccabscdab», при этом используется только начальный словарь (табл. 1).

ИНДЕКС	ВХОД
0	a
1	b
2	c
3	d

Табл. 1 – Начальный словарь

Этапы кодирования будут выполняться следующим образом (табл. 2).

ВХОД	ТЕКУЩАЯ СТРОКА	ВИДЕЛИ ЭТО РАНЬШЕ?	КОДИРОВАННЫЙ ВЫХОД	НОВАЯ ЗАПИСЬ В СЛОВАРЕ/ИНДЕКС
<i>a</i>	<i>a</i>	да	ничего	никакой
<i>ab</i>	<i>ab</i>	нет	0	<i>ab</i> / 4
<i>abd</i>	<i>bd</i>	нет	0, 1	<i>bd</i> / 5
<i>abda</i>	<i>da</i>	нет	0, 1, 3	<i>da</i> / 6
<i>abdab</i>	<i>ab</i>	да	без изменений	никакой
<i>abdabc</i>	<i>abc</i>	нет	0, 1, 3, 4	<i>abc</i> / 7
<i>abdabcc</i>	<i>cc</i>	нет	0, 1, 3, 4, 2	<i>cc</i> / 8
<i>abdabcca</i>	<i>ca</i>	нет	0, 1, 3, 4, 2, 2	<i>ca</i> / 9
<i>abdabccab</i>	<i>ab</i>	да	без изменений	никакой
<i>abdabccabc</i>	<i>abc</i>	да	без изменений	никакой
<i>abdabccabcd</i>	<i>abcd</i>	нет	0, 1, 3, 4, 2, 2, 7	<i>abcd</i> / 10
<i>abdabccabceda</i>	<i>da</i>	да	без изменений	никакой
<i>abdabccabcedab</i>	<i>dab</i>	нет	0, 1, 3, 4, 2, 2, 7, 6	<i>dab</i> / 11
<i>abdabccabcedab</i>	<i>b</i>	да	0, 1, 3, 4, 2, 2, 7, 6, 1	никакой

Табл. 2 – Процесс кодирования при входной строке «abdabccabcedab»

После считывания последнего символа, ‘b’, должен быть выведен индекс последней подстроки, ‘b’.

### 3.2. Псевдокод алгоритма сжатия

```

string s;
char ch;
...

s = empty string; //пустая цепочка
while (there is still data to be read) //пока есть символы во входном потоке
{
    ch = read a character; //взять очередной символ
    if (dictionary contains s+ch) //цепочка s+ch уже есть в таблице
    {
        s = s+ch; //удлиняем цепочку прочитанным символом
    }
}

```

```

else //цепочки s+ch нет в таблице
{
    encode s to output file; //вывод кода, соответствующего цепочке s
    add s+ch to dictionary; //добавляем новую цепочку s+ch в таблицу
    s = ch; //готовимся к «обнаружению» следующей цепочки
}
}
encode s to output file; //вывод кода для оставшейся цепочки s

```

Алгоритм кодирования каждый раз пытается найти в таблице наиболее длинную цепочку, соответствующую читаемой последовательности символов. Если это в какой-то момент не удастся, то накопленная к этому времени цепочка заносится в таблицу. В какой-то момент может наступить переполнение таблицы. В этом случае кодировщик выводит в выходной поток специальный код очистки, и таблица цепочек инициализируется заново. Обычно в реальных алгоритмах применяется таблица с 4096 входами для цепочек, при этом число 256 является кодом очистки (CLC), а число 257 – кодом конца информации (EOI), эти строки таблицы не используются для цепочек.

Заметим, что в этом случае для каждого выходного кода достаточно 12 бит. Поэтому при выводе целесообразно упаковывать каждые два кода в три байта.

## 4. Распаковка

Формирование словаря независимо от того, находимся мы на стороне кодировщика или декодировщика. И кодировщик и декодировщик работают по одной и той же схеме, и поэтому словарь они заполняют как бы синхронно. Поэтому словари у них получаются идентичные и никакой дополнительной информации о частотных или каких-либо ещё характеристиках набора символов им передавать не нужно. В процессе работы словарь, идентичный тому, который был создан во время сжатия, восстанавливается. Программы кодирования и декодирования должны начинаться с одного и того же начального словаря.

Декодер LZW сначала считывает индекс (целое число), ищет этот индекс в словаре и выводит подстроку, связанную с этим индексом. Первый символ этой подстроки конкатенируется с текущей рабочей строкой. Эта новая конкатенация добавляется в словарь. Затем декодированная строка становится текущей рабочей строкой (текущий индекс, т. е. подстрока, запоминается), и процесс повторяется.

### 4.1. Пример

Этапы декодирования будут выполняться следующим образом (табл. 3) (при использовании всё того же начального словаря из табл. 1.).

КОДИРОВАННЫЙ ВХОД	ПРЕОБРАЗОВАНИЕ СЛОВАРЯ	ДЕКОДИРОВАННЫЙ ВЫХОД	ТЕКУЩАЯ СТРОКА	НОВАЯ СЛОВАРНАЯ ЗАПИСЬ/ИНДЕКС
0	$0 = a$	<i>a</i>	никакая	никакая
0, 1	$1 = b$	<i>a</i> <u><i>b</i></u>	<i>a</i>	<i>ab</i> / 4
0, 1, 3	$3 = d$	<i>ab</i> <u><i>d</i></u>	<i>b</i>	<i>bd</i> / 5
0, 1, 3, 4	$4 = ab$	<i>abd</i> <u><i>ab</i></u>	<i>d</i>	<i>da</i> / 6
0, 1, 3, 4, 2	$2 = c$	<i>abdab</i> <u><i>c</i></u>	<i>ab</i>	<i>abc</i> / 7
0, 1, 3, 4, 2, 2	$2 = c$	<i>abdabcc</i> <u><i>c</i></u>	<i>c</i>	<i>cc</i> / 8
0, 1, 3, 4, 2, 2, 7	$7 = abc$	<i>abdabcc</i> <u><i>abc</i></u>	<i>c</i>	<i>ca</i> / 9
0, 1, 3, 4, 2, 2, 7, 6	$6 = da$	<i>abdabccabc</i> <u><i>da</i></u>	<i>abc</i>	<i>abcd</i> / 10



0, 1, 3, 4, 2, 2, 7, 6, 1	$1 = b$	abdabccabcbda <u>b</u>	da	dab / 11
---------------------------	---------	------------------------	----	----------

Табл. 3 – Процесс декодирования при кодированном входе «0, 1, 3, 4, 2, 2, 7, 6, 1»

## 4.2. Возможные проблемы при декодировании

Есть исключение, когда алгоритм не работает; это происходит, когда код вызывает индекс, который еще не был введен в словарь (или когда только что закодированная строка встречается на входе). Например, предположим, у нас есть строка «abababa» и начальный словарь, состоящий только из а и b с индексами 0 и 1 соответственно. Тогда процесс кодирования будет состоять из следующих этапов (табл. 4).

ВХОД	ТЕКУЩАЯ СТРОКА	ВИДЕЛИ ЭТО РАНЬШЕ?	КОДИРОВАННЫЙ ВЫХОД	НОВАЯ ЗАПИСЬ В СЛОВАРЕ/ИНДЕКС
<i>a</i>	<i>a</i>	да	ничего	никакой
<i>ab</i>	<i>ab</i>	нет	0	ab / 2
<i>aba</i>	<i>ba</i>	нет	0, 1	ba / 3
<i>abab</i>	<i>ab</i>	да	без изменений	никакой
<i>ababa</i>	<i>aba</i>	нет	0, 1, 2	aba / 4
<i>ababab</i>	<i>ab</i>	да	без изменений	никакой
<i>abababa</i>	<i>aba</i>	да	0, 1, 2, 4	никакой

Табл. 4 – Процесс кодирования при входной строке «abababa»

Тогда при декодировании возникает следующая проблема (табл. 5).

КОДИРОВАННЫЙ ВХОД	ПРЕОБРАЗОВАНИЕ СЛОВАРЯ	ДЕКОДИРОВАННЫЙ ВЫХОД	ТЕКУЩАЯ СТРОКА	НОВАЯ СЛОВАРНАЯ ЗАПИСЬ/ИНДЕКС
0	$0 = a$	<i>a</i>	никакая	никакая
0, 1	$1 = b$	<i>a<u>b</u></i>	a	ab / 2
0, 1, 2	$2 = ab$	<i>ab<u>ab</u></i>	b	ba / 3
0, 1, 2, 4	$4 = ???$	<i>abab???</i>	ab	???

Табл. 5 – Процесс декодирования при кодированном входе «0, 1, 2, 4»

Таким образом, чтобы алгоритм справился с этим исключением, нужно взять подстроку, которую уже получили, 'ab', и конкатенировать ее первый символ с самой собой, 'ab' + 'a' = "aba" вместо того, чтобы следовать процедуре как обычно.

#### 4.3. Псевдокод алгоритма распаковки

```
while ((k = nextcode)) != EOI) //пока не конец информации
{
    if (k == CLC) //k есть код очистки
    {
        InitTable(); //заново инициализируем таблицу
        k = nextcode; //читаем следующий код
        if (k == EOI)
            break;
        OutString(k); //выводим цепочку для кода k
        old = k; //запоминаем текущий код
    }
    else
    {
        if (InTable(k)) //в таблице есть строка для кода k
        {
            OutString(k); //выводим цепочку для кода k
            AddString(String(old)+Char(k)); //формируем и добавляем новую цепочку
            old = k;
        }
        else // в таблице нет строки для кода k
        {
            s = String(old)+Char(old); //формируем цепочку
            OutString(s); //выводим цепочку
            AddString(s); //и добавляем её в таблицу
            old = k;
        }
    }
}
```

## 5. Оценка алгоритма

Будем считать, что базовый алфавит закодирован числами от 0 до  $2^n - 1$  записанными  $n$ -битными блоками, и кодирование собственно документа начинается с номера  $2^n$ . В таких предположениях, количество бит, необходимых для кодирования  $L$  первых слов кодируемого текста равно

$$\sum_{j=n}^{K-1} 2^j(j+1) + (L - \sum_{j=n}^{K-1} 2^j)(K+1) = (K-n+2)2^n - 2^{K+1} + L(K+1)$$

при  $K = \lfloor \log^2 L \rfloor$  (  $\lfloor \cdot \rfloor$  означает целую часть числа ).

К примеру текст, содержащий 613 символов и с исходным словарём в 3 символа, может быть закодирован  $3 \cdot 613 = 1839$  битами. LZW алгоритм в таком случае даёт 241 цепочку в словарь. Стоимость кодирования, при использовании алгоритма, составит  $3 \cdot 3 + 8 \cdot 4 + 16 \cdot 5 + 32 \cdot 6 + 64 \cdot 7 + (241 - 3 - 120) \cdot 8 = 1705$  бит.  $1705/1839 \approx 0.927$ , из этого следует, что эффективность сжатия составит порядка 10%.

LZW-сжатие выделяется среди прочих, когда встречается с потоком данных, содержащим повторяющиеся строки любой структуры. По этой причине он работает весьма эффективно, когда встречает английский текст. Уровень сжатия может достигать 50% и выше.

Поскольку размер словаря фиксирован и не зависит от длины ввода, сложность LZW -  $O(n)$ , т. к. каждый байт считывается только один раз, а сложность операции для каждого символа постоянна. Хранение словаря таким образом, вероятно, отнимает много памяти, так что это компромисс между скоростью и пространством, а эффективная реализация памяти, вероятно, будет составлять по крайней мере  $O(n \log n)$ .

## **6. Тесты**

Ниже приведены тесты, используемые для автоматического тестирования, итоговой реализации.

### **Тест 1**

Проверка на пустоту файла.

### **Тест 2**

Проверка на отсутствие данных для кодирования.

### **Тест 3**

Проверка на отсутствие данных для декодирования.

### **Тест 4**

Проверка на незначащие символы (пробелы, переносы строк).

### **Тест 5**

Проверка на незначащие символы при декодировании (пробелы, переносы строк).

### **Тест 6**

Проверка на тип входных данных при декодировании.

### **Тест 7**

Проверка на существование записи в словаре при начальном заходе в алгоритм декодирования.

### **Тест 8**

Проверка на запрос encode/decode.

### **Тест 9**

Проверка на подряд идущие пробелы/переносы строк при декодировании.

### **Тест 10–13**

Проверка на корректность работы кодирования.

## **Тест 14–17**

Проверка на корректность работы декодирования.

## Список литературы

1. Семенюк В. В. Экономное кодирование дискретной информации. –СПб.: СПбГИТМО (ТУ), 2001. – 115 с. - Режим доступа: [http://compression.ru/download/articles/rev\\_univ/semenyuk\\_2001\\_econom\\_en\\_coding.pdf](http://compression.ru/download/articles/rev_univ/semenyuk_2001_econom_en_coding.pdf)
2. Mark R. Nelson «LZW Data Compression» [Электронный ресурс]: электрон. статья/ Mark R. Nelson, 1989. – Режим доступа: <https://marknelson.us/posts/1989/10/01/lzw-data-compression.html>
3. Алгоритм LZW [Электронный ресурс]: электрон. статья/ информационный ресурс ИТМО, 2022. – Режим доступа: [http://neerc.ifmo.ru/mediawiki/index.php?title=Алгоритм\\_LZW&redirect=no](http://neerc.ifmo.ru/mediawiki/index.php?title=Алгоритм_LZW&redirect=no)
4. Hans Wennborg «Shrink, Reduce, and Implode: The Legacy Zip Compression Methods» [Электронный ресурс]: электрон. статья/ Hans Wennborg, 2021. – Режим доступа: <https://www.hanshq.net/zip2.html>
5. «Подстановочные или словарно-ориентированные алгоритмы сжатия информации. Методы Лемпела-Зива» [Электронный ресурс]/ ИНТУИТ, 2007. Режим доступа: <https://intuit.ru/studies/courses/2256/140/lecture/3904>
6. Michael Dipperstein «Lempel-Ziv-Welch (LZW) Encoding Discussion and Implementation» [Электронный ресурс]: электрон. статья/ Michael Dipperstein, 2015. Режим доступа: <http://michael.dipperstein.com/lzw/>
7. Ziv J., Lempel A. «Compression of Individual Sequences via Variable-Rate Coding. » / IEEE Trans. Inform. Theory, 1978 – 530-536 с
8. Welch T. «A Technique for High-Performance Data Compression. »/ Computer, 1984 – 8-19 с.
9. Сэломон Д. «Сжатие данных, изображений и звука.» / М.Техносфера. 2006.
10. Пантелеев Е. Р., Алыкова А. Л. «Алгоритмы сжатия данных без потерь»: учебное пособие для вузов / Е. Р. Пантелеев, А. Л. Алыкова. — Санкт-Петербург: Лань, 2021. – 172 с