МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Дальневосточный федеральный университет»
(ДВФУ)

Алгоритм Лемпеля-Зива-Велча (LZW)

Выполнил студент:
гр. №Б9121 – 09.03.03пикд
Кирилов Денис Юрьевич

г. Владивосток
2023 г.

ВВЕДЕНИЕ

- LZW (Lempel-Ziv-Welch) алгоритм - это метод сжатия данных, который использует кодирование словарем для замены повторяющихся последовательностей данных на их соответствующие индексы.
- Оригинальная версия метода была разработана Авраамом Лемпелем и Яковом Зивом в 1978 году (LZ78) и доработана Терри Уэлчем в 1984 году.
- LZW использует основанный на словаре подход для сжатия данных. Он работает путем замены повторяющихся подстрок в данных уникальным кодом, соответствующим индексу этой подстроки в словаре, что уменьшает общий размер данных.

Алгоритм LZW работает следующим образом:

1. Создается словарь, который содержит все отдельные символы исходного текста и их коды (индексы).
2. Проходится по исходному тексту, и каждая последовательность символов заменяется на соответствующий ей код из словаря.
3. В словарь добавляются новые сочетания символов, если они встречаются в исходном тексте и отсутствуют в словаре.

ПРЕИМУЩЕСТВА И НЕДОСТАТКИ

ПРЕИМУЩЕСТВА АЛГОРИТМА

- Алгоритм является однопроходным.
- Для декомпрессии не надо сохранять таблицу строк в файл для распаковки
- Высокая степень сжатия
- Простота реализации
- Достаточно высокая скорость сжатия и распаковки.

НЕДОСТАТКИ АЛГОРИТМА

- Ограничение размером словаря

ПРИМЕНЕНИЕ

ОСНОВНЫЕ СФЕРЫ ПРИМЕНЕНИЯ АЛГОРИТМА LZW:

LZW алгоритм может применяется для:

- Сжатия изображений и видео.
- Сжатия текста.
- Сжатия исполняемых программ.
- Сжатия потоков данных.
- Сжатия аудиосигналов.
- Сжатия медицинских изображений.
- Сжатия спутниковых изображений.
- Сжатия веб-графики.
- Сжатия файлов автоматизированного проектирования.

СЖАТИЕ

Основные шаги алгоритма сжатия LZW заключаются в следующем:

1. Словарь инициализируется всеми возможными односимвольными цепочками с уникальными индексами.
2. Входные данные считываются по одному символу за раз и для каждого символа алгоритм проверяет, образуют ли предыдущие символы плюс текущий символ новую подстроку.
3. Если найдена новая подстрока, она заносится в словарь с новым уникальным кодом. Выводится код для предыдущей подстроки.
4. Если текущая подстрока уже встречалась, считывается следующий символ.
5. Шаги 2-4 повторяются до тех пор, пока не будут обработаны все входные данные.
6. Выводится окончательный код для последнего встреченной подстроки.

ПРИМЕР СЖАТИЯ

Предположим, что наш поток, который мы хотим сжать, это «abdabccabcdab», при этом используется только начальный словарь **a/0**, **b/1**, **c/2**, **d/3**.

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

ПСЕВДОКОД

```
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
```

РАСПАКОВКА

Во время распаковки алгоритм использует словарь для замены кодов исходными подстроками.

1. Декодер LZW сначала считывает индекс (целое число) и ищет этот индекс в словаре.
2. Выводится подстрока, связанную с этим индексом.
3. Первый символ этой подстроки конкатенируется с текущей рабочей строкой.
4. Новая конкатенация добавляется в словарь.
5. Декодированная строка становится текущей рабочей строкой (текущий индекс, т. е. подстрока, запоминается).
6. Процесс повторяется пока не будут обработаны все входные данные.

ПРИМЕР РАСПАКОВКИ

Предположим, что наш поток, который мы хотим декодировать, это «**0, 1, 3, 4, 2, 2, 7, 6, 1**», при этом используется тот же начальный словарь **a/0, b/1, c/2, d/3.**

КОДИРОВАННЫЙ ВХОД	ПРЕОБРАЗОВАНИЕ СЛОВАРЯ	ДЕКОДИРОВАННЫЙ ВЫХОД	ТЕКУЩАЯ СТРОКА	НОВАЯ СЛОВАРНАЯ ЗАПИСЬ/ ИНДЕКС
0	0 = a	a	никакая	никакая
0, 1	1 = b	ab	a	ab / 4
0, 1, 3	3 = d	abd	b	bd / 5
0, 1, 3, 4	4 = ab	abdab	d	da / 6
0, 1, 3, 4, 2	2 = c	abdabc	ab	abc / 7
0, 1, 3, 4, 2, 2	2 = c	abdabcc	c	cc / 8
0, 1, 3, 4, 2, 2, 7	7 = abc	abdabccabc	c	ca / 9
0, 1, 3, 4, 2, 2, 7, 6	6 = da	abdabccabcda	abc	abcd / 10
0, 1, 3, 4, 2, 2, 7, 6, 1	1 = b	abdabccabcdab	da	dab / 11

ПСЕВДОКОД

```
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;
   }
   …
```

АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ

Поскольку размер словаря фиксирован и не зависит от длины ввода. Каждый байт считывается только один раз, а сложность операции для каждого символа постоянна, то

- Сложность LZW равна **O(n),** (где n - количество символов во входных данных)

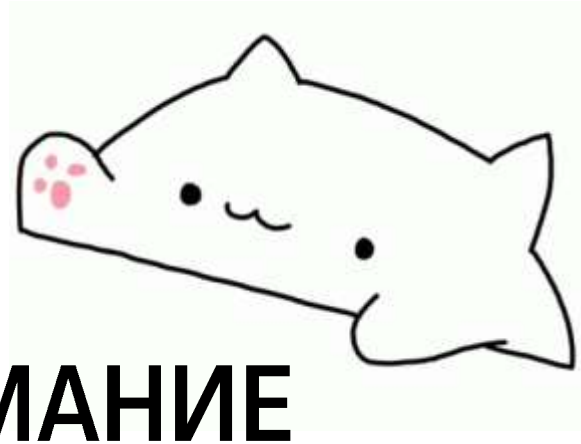
Время выполнения алгоритма линейно увеличивается с размером входных данных.

- Пространственная сложность алгоритма LZW обычно равна **O(k),** где k - количество уникальных строк во входных данных.

Объем памяти, требуемый алгоритмом, увеличивается с увеличением количества уникальных строк во входных данных.

ОПИСАНИЕ РЕАЛИЗАЦИИ

- Язык программирования – C++
- Описаны основные функции алгоритма LZW (считывание данных, кодирование (сжатие), декодирование (распаковка))
- Текстовый тип входных данных
- Для выбора функции алгоритма в первой строке входных данных требуется вписать **encode** – для кодирования или **decode** – для декодирования
- Поэтапное выполнение (считывание, сравнение со словарём, заполнение словарей и/или переход к следующей цепочке и т. д.)



СПАСИБО ЗА ВНИМАНИЕ