Компактная реализация модифицированного алгоритма Лемпеля—Зива—Велча

А.Л. Петрунёв, <alexeypetrunev@gmail.com>

Г.А. Ситкарев, <sitkarev@unixkomi.ru>

Сыктывкарский Государственный Университет
Лаборатория Прикладной Математики и Программирования
http://amplab.syktsu.ru

РЕФЕРАТ

Приводится краткое описание реализации модифицированного алгоритма LZW. Даётся описание используемого метода и библиотеки потокового сжатия без потерь liblzw.

1. Введение

Библиотека предоставляет реализацию модифицированного универсального алгоритма сжатия данных без потерь LZW (Lempel–Ziev–Welch), опубликованного в 1984 г. в [1] и ранее в [2].

В отличие от большинства реализаций алгоритмов сжатия, ориентированных в первую очередь на файловый ввод–вывод, данная библиотека предоставляет программисту лишь сам механизм, не накладывая никаких ограничений на его применение. Сжатие данных может выполняться в потоковом режиме по мере поступления данных, что позволяет использовать библиотеку как часть прикладных протоколов межсетевого обмена.

В библиотеке присутствует две группы функций: функции сжатия и разжатия. В оригинальном алгоритме 8-битные значения кодируются 12-битными кодами фиксированной длины. При этом коды от 0 до 255 обозначают соответствующие однобайтовые значения, а коды от 256 до 4095 добавляются в словарь во время работы алгоритма. Каждый входной байт добавляется в буфер, пока не будет получен такой байт, что для последовательности, собранной в буфере, не окажется кода в словаре. Тогда на выход выдаётся код последовательности (без последнего байта), а последовательность (включая последний байт) добавляется в словарь и получает следующий по порядку код.

Если длина кода фиксирована, то в начальном состоянии алгоритма, когда словарь ещё не заполнен, его эффективность существенно снижается. Потому практически целесообразно принять длину кода переменной, и увеличивать её до определённого максимума (обычно 12 бит) по мере заполнения словаря. Начальную длину кода обычно берут на один бит больше, чем это нужно для кодирования однобайтовых последовательностей.

Часть кодов (обычно начиная с 256) резервируется для специальных целей. Один из кодов назначается для сброса словаря («clear code»), при получении которого алгоритм разжатия должен сбросить словарь в начальное состояние; далее словарь заполняется заново.

Некоторые реализации алгоритмов отслеживают эффективность сжатия и сбрасывают словарь в начальное состояние, если эффективность снижается. Таким образом осуществляется адаптация к входным данным. Если словарь был сброшен, то на выход выдаётся код сброса словаря. Ещё один код выделяется для обозначения конца данных («stop code»). Этот код указывает алгоритму разжатия, что больше данных посылаться не будет.

Словарь с кодами последовательностей формируется по мере поступления данных. Поэтому алгоритм разжатия формирует словарь, имитируя алгоритм сжатия. Важно, чтобы соглашения (порядок байт, размер словаря, начальный код и длины кодов), принятые на стороне алгоритма сжатия, совпадали с соглашениями, принятыми на стороне алгоритма разжатия.

Высокоуровневое описание алгоритма

Не вдаваясь в подробности реализации, алгоритм сжатия выполняет следующие шаги:

1. Словарь заполняется однобайтовыми последовательностями с кодами 0–255.
2. В словаре находят самую длинную строку S, которая совпадает с текущим вводом.
3. На выход выдаётся код S, взятый из словаря, а с ввода последовательность S убирают.
4. В словарь добавляют последовательность S плюс следующий байт ввода.
5. Переходят на шаг 2.

В начале словарь заполняют всеми возможными однобайтовыми последовательностями. Затем алгоритм начинает искать в словаре последовательности всё большей и большей длины, совпадающие с вводом, пока наконец не будет найдена последовательность, совпадения для которой в словаре нет. Тогда на выход выдаётся код для этой последовательности, исключая последний байт (эта последовательность гарантированно есть в словаре), а сама последовательность, включая последний байт, добавляется в словарь и ей присваивается следующий по порядку код. Последний байт становится начальным символом для следующего поиска входной последовательности.

Таким образом, словарь заполняется строками возрастающей длины, которые представляются на выходе одним кодовым значением. Алгоритм хорошо работает с данными, где имеются повторяющиеся последовательности. Начальная порция входных данных сжимается слабо, так как словарь ещё не заполнен. По мере заполнения словаря, эффективность алгоритма сжатия растёт.

Алгоритм разжатия выполняет следующие шаги:

1. Словарь заполняется однобайтовыми последовательностями с кодами 0–255.
2. Из входного потока бит извлекается значение кода.
3. В словаре выполняется поиск кода, и на выход выдаётся последовательность, соответствующая ему.
4. В словарь добавляется строка, состоящая из предыдущей выходной последовательности, плюс первый байт текущей выходной последовательности, и ей присваивается следующий по порядку код.
5. Текущая выходная последовательность запоминается.
6. Переходят на шаг 2.

Таким образом, алгоритм разжатия сформирует тот же самый словарь, что и алгоритм сжатия. Сам словарь поэтому передавать нет необходимости, а его начальное состояние известно обеим сторонам.

Коды переменной длины

Если используются коды переменной длины, то алгоритмы сжатия и разжатия должны увеличивать длину кода согласованно, иначе произойдёт рассинхронизация, и восстановить сжатые данные будет невозможно. Алгоритм сжатия увеличивает длину кода из n в n+1 тогда, когда для последовательности <S,c> нет записи в словаре (а значит её нужно добавить в словарь), а следующий по счёту код это 2^n (наименьший код, для представления которого нужно n+1 бит). При этом на выход выдаётся код для S с длиной n, а последующие коды будут иметь длину n+1.

Алгоритм разжатия всегда отстаёт на один код при заполнении словаря. Поэтому длину кода он увеличивает с n до n+1 тогда, когда он получит код для последовательности S и добавит новую последовательность в словарь с кодом 2^{n-1}. Все последующие коды, извлекаемые алгоритмом из входного потока бит, будут иметь длину n+1.

Когда алгоритм сжатия выдал код очистки словаря, то алгоритм разжатия должен по его получении очистить словарь, инициализировать его, и сбросить длину кода на начальную. Значение следующего кода словаря устанавливается на значение наибольшего управляющего кода плюс 1.

Порядок упаковки

Длина кода (в битах) больше длины байта и не кратна его длине, поэтому при упаковке коды не попадают точно на границу байта. В связи с этим, алгоритмы сжатия и разжатия должны согласовать порядок упаковки кодов в байты. В данной библиотеке принято соглашение «старший бит идёт первым». Таким образом старшие биты кода упаковываются в первый байт, а оставшиеся младшие биты кода в следующий байт.

2. Практическая реализация

Библиотека, реализующая алгоритм, написана на языке Си, отличается высокой переносимостью и хорошим быстродействием. На классическом тесте Calgary [3] библиотека показала коэффициент сжатия 2,1. Тексты русской классики при сжатии алгоритмом также показывали коэффициент ≈ 2, например роман «Война и мир» Л.Н. Толстого в формате TXT до сжатия имел размер 1444709 байт, а после сжатия 793362 байт.

Код библиотеки не требует никакого специфичного окружения времени выполнения, не выделяет память в куче, не использует динамические аллокаторы памяти. Размер структур данных для хранения контекстов сжатия и разжатия соответственно 35 Кбайт и 15 Кбайт. Портировать код библиотеки на 16-разрядные архитектуры или микроконтроллеры не представляет особого труда. Исходные тексты библиотеки доступны на условиях BSD лицензии с репозитория subversion по адресу [4], или для просмотра через Web по адресу [5].

3. Заключение

Реализация алгоритма готова к практическому использованию в составе различных протоколов или специализированных систем связи и способна повысить пропускную способность каналов связи до двух раз. Эвристические правила и нечёткая логика для специальных типов сжимаемых данных могут улучшить степень сжатия алгоритма, манипулируя сбросом словаря в подходящие для этого промежутки, однако при этом снизят скорость сжатия.

Ссылки

[1] Welch, T.A., «A Technique for High-Performance Data Compression», *IEEE Computer, no. 6, vol. 17, pp. 8-19, June 1984.*

[2] Ziv, J. and Lempel, A., «Compression of Individual Sequences via Variable-Rate Coding», *IEEE Transactions on Information Theory, no. 5, vol. IT-24, pp. 530-536, September 1978.*

[3] http//compression.ca/act/act-calgary.html

[4] http://wiki.syktsu.ru/svn/libslzw

[5] http://wiki.syktsu.ru/websvn/listing.php?repname=libslzw&path=%2F&sc=0