УДК 004.021
В.Ю. Василенко, Р.С. Самарев

Способы реализации и ускорения алгоритма сжатия LZ77

Введение

В настоящее время широкое распространение получили алгоритмы сжатия без потерь, такие как семейство алгоритмов Зива-Лемпеля. Алгоритмы LZ77, LZ78 стали основой для многих более эффективных алгоритмов – LZMA (применяется в 7-Zip архиваторе), LZW (применяется в GIF), DEFLATE (используется в утилитах pkzip, gzip, реализован в библиотеке Zlib). Известные алгоритмы сжатия без потерь отличаются по множеству характеристик, основные из которых это - эффективность сжатия, скорость сжатия/распаковки. Скорость зависит от: эффективности реализации алгоритма, природы исходных данных, используемых структур и методов поиска фраз в словаре. Анализ существующих алгоритмов позволяет оценить их работу на конкретном типе данных, узнать скорость и эффективность алгоритмов сжатия.

Описание алгоритма LZ77 и его модификации LZSS

Алгоритм LZ77 и его различные модификации называют алгоритмами со скользящим окном. Скользящее окно представляет собой блок данных, прошедших кодирование – словарь, и буфер содержащий фразу, которая ищется в словаре. Когда окно скользит, по входным данным, в словарь из буфера добавляются обработанные последовательности символов в конец, и затираются старые последовательности, находящиеся в начале словаря.

Формальное описание алгоритма LZ77:

Пусть есть окно длиной N состоящее из двух частей:

- Последовательности длиной W = N-n уже закодированных символов, словарь.
- Буфера предварительного просмотра длиной n, причем n << N

Пусть уже закодировано t символов S1,S2,…,St, соответственно словарь состоит из этих t символов, а буфер содержит St+1, St+2,…,Sn символы. Идея алгоритма состоит в нахождении самой длинной фразы в словаре, совпадающей с буфером. В результате поиска найдена фраза в словаре: SN-i+1, SN-i+2,…SN-i+k символы которой совпадают с символами буфера St+1, St+2,…,St+k, где k количество совпавших символов словаря и буфера, i – смещение относительно начала буфера N+1, причем i>0; k>0. Тогда фразу из буфера длиной k можно закодировать с помощью двух чисел:

1) Смещение от начала буфера i
2) Длинны совпадения k

В случае если не была найдена совпадающая фраза в словаре k=0, смещение i кодируется как 0 и k как 1, после чего записывается первый символ фразы в словаре на позиции N+1.

Так как возникает избыточность при записи кодов, которые так же записываются и в случае, если искомая фраза не была найдена в словаре, имеет смысл использовать модификацию алгоритма LZSS. Алгоритм LZSS аналогичен в принципе поиска совпадающей фразы, но все кодируемые фразы записываются с бит-флагом спереди, разделяющим коды на отдельные символы и найденные фразы словаря. Кодируются фразы следующим образом:

-если k>0, т.е. в словаре найдена совпадающая фраза, записываем 1 бит равный единице, и смещение и длину совпадения

-если k=0 т.е. в словаре не найден совпадающая фраза, записывает 1 бит равный нулю, и первый символ буфера.

Алгоритм распаковки сводится к последовательному чтению кодов и заполнению окна. Окно формируется аналогично упаковке, но исходные позиции фраз буфера уже даны в кодах. Причем для LZSS необходимо считывать один бит-флаг, указывающий тип кода: фраза или символ.

Организация окна

Окно можно реализовать множеством способов. Одни из этих способов – это организация в виде буфера, с указателями и в виде замкнутой очереди, который эффективен тем, что не требует проведения операций копирования памяти, в данном методе вся выделенная память расходуется по назначению, то есть заполняется данными словаря.

Реализация в виде буфера представляет собой буфер величины P, причем P>N, и указатели на начало окна pW и на начало буфера pB предварительного просмотра. В процессе скольжения окна указатели перемещаются по буферу на величину k для каждой фразы. Для более подробного ознакомления смотрите Рис.1.

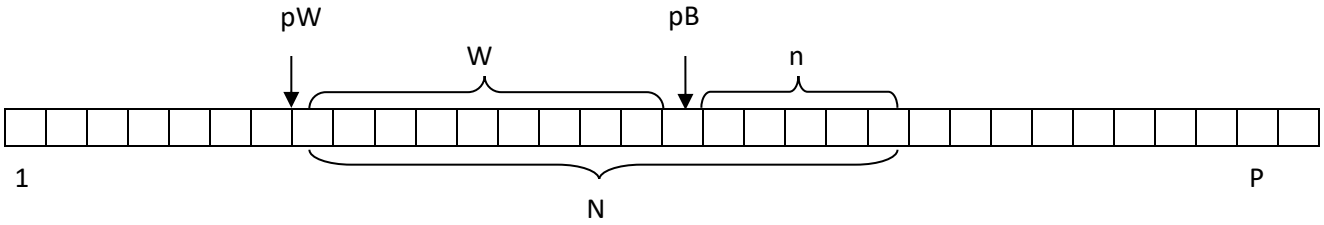


Рисунок 1 – Структура скользящего окна для реализации в виде буфера

При достижении конца буфера, когда pB = P-n+1, в буфер загружаются новые данные и указатели устанавливаются на начало буфера. В начало буфера нужно скопировать окно, чтобы данные о фразах из предыдущей порции данных не были утеряны – словарь и упреждающий буфер. Плюсы данной схемы – это простота реализации, удобное перемещение окна по буфер, т.к. оно фактически по настоящему скользит вдоль данных. Минусы этого способа в требовании дополнительной памяти под сам буфер, который должны быть в несколько раз больше окна, и в необходимости копирования данных окна в начало при достижении им конца буфера, на что тратиться время.

Реализация в виде замкнутой очереди. Эта реализация предполагает, что мы выделяем буфер соответствующего размера под окно. В буфер окна записываются данные. При добавлении нового символа конец очереди сдвигается на одну позицию. Начало окна двигается только в случае, если окно заполнилось, то есть выполняется условие count = W, где count размер очереди. Конец окна двигается постоянно при добавлении данных, и указывает на следующий индекс для записи символа. Изменение позиций начала и конца окна - front и back задается таким образом, чтобы эти индексы не могли выйти за пределы буфера, для чего берется модуль, получаем, что front = (front+1) mod W и back = (back+1) mod W. Операцию остатка от деления - mod можно заменить на побитовую “И” если вместо W брать битовую маску по размеру окна, где MASK = W-1, но размер окна в данном случае должен быть кратен 2-ке. Такой способ представления окна решает проблему в потребности дополнительной памяти, а также избавляет от необходимости копирования данных окна из одного участка буфера в другой.

Методы поиска фраз в словаре

Существует множество способов поиска, рассмотрим самый очевидный – наивный метод поиска, и один из самых быстрых – хэш-поиск.

Наивный метод.

Основан на поиске буфера предварительного просмотра подобно поиску подстроки в словаре. Поиск может осуществляться посимвольным сравнением и движением по словарю в буфере. Из множества совпадений выбирается наибольшее. Наивный метод можно ускорить применив алгоритм Бойера-Мура (или его различные модификации) для сдвига плохого символа, но это не спасает от его низкой производительности, т.к. приходится для каждого экземпляра буфера предварительного просмотра проходить весь словарь и сравнивать его целиком с образцом, по всему размеру буфера, это очень медленная операция, особенно если словарь 1 Мб, 4 Мб и больше.

Хэш-поиск.

Использует индексацию данных, с помощью некоторой хэш-функции (Хэш-функцию применяемую при сжатии данных смотри в [1]). По данным словаря составляется хэш-таблица, где индексами являются значения хэш-функции для определенных фраз словаря, а значениями самой таблицы являются позиции фраз в словаре соответствующие их индексам. Для реализации данного метода поиска кроме самой таблицы необходим список хэш-цепочек, которые содержат одинаковые значения хэш-функции для различных данных. Попадая через хэш-таблицу на одну из хэш-цепочек ведется поиск значения соотвествующего началу фразы совпадения (т.е. заданное данным поиска). Из найденных фраз выбирается самая длинная, которая позволит закодировать большее число символов.

Для ускорения хэш-поиска можно применить ограничение на глубину поиска. Такой прием позволяет существенно сократить время поиска лучше совпадения. А потери в данном случае не столь велики, так как хэш-цепочки могут становиться очень длинными: при малом размере хэш-таблицы, плохом распределении хэш-значений для диапазона возможных данных, сильной однородности данных. В таком случае близкие к лучшим совпадениям могут быть найдены в самом начале цепочки. Конечно потери в качестве сжатия могут быть и существенными, но для большинства пользователей скорость может играть большую роль. Поэтому глубина поиска должна быть параметром в реализации алгоритма, и ее значение устанавливает сам пользователь.

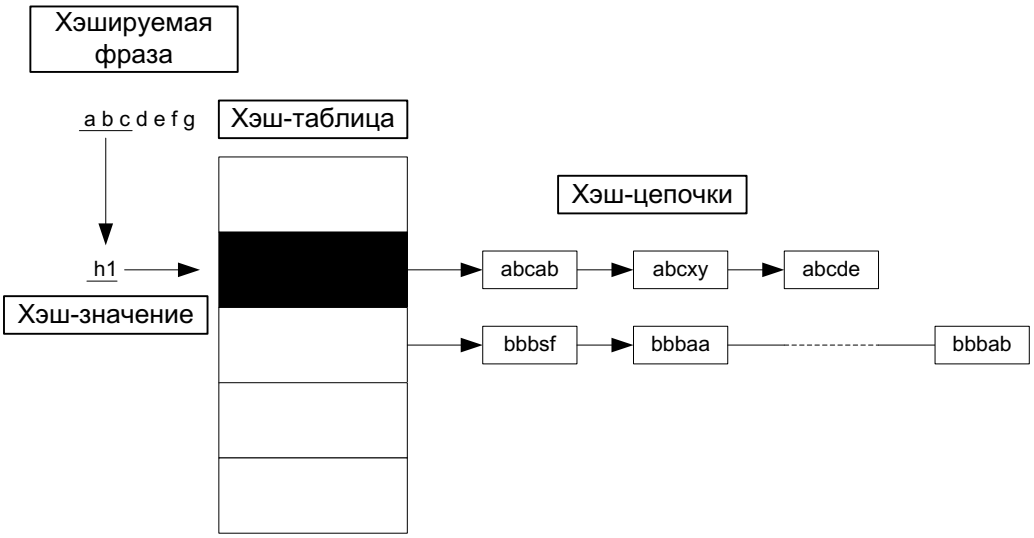


Рисунок 2 – Процесс хэширования строки

На рисунке 2 изображен процесс хэширования строки “abcdefg”. Каждая строка хэшируется по первым трем символам. Из первых трех символов получаем с помощью хэш-функции f1 значение h1. В таблицу по индексу h1 заносится положение строки в словаре. Т.к. одному значению h1 может соответствовать множество строк из 3 символов, положение строки добавляется в начало хэш-цепочки. Таким образом если необходимо найти строку “abcdefg” в словаре достаточно вычислить её хэш-значение и взяв его за индекс обратится к соответствующему значению в хэш-таблице положения строки, если строка в заданной позиции не соответствует исходной строке требуется выполнить поиск по хэш-цепочке данного значения, пока не будет найдена искомая строка.

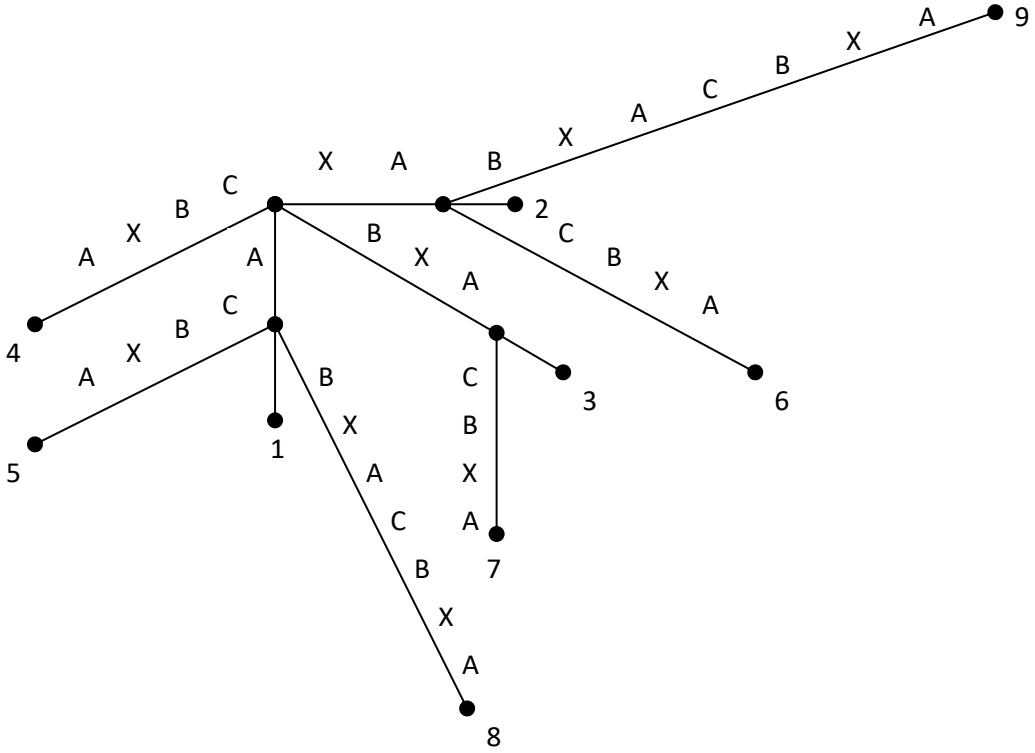


Рисунок 3 – Суффиксальное дерево для строки “XABXACBXA”

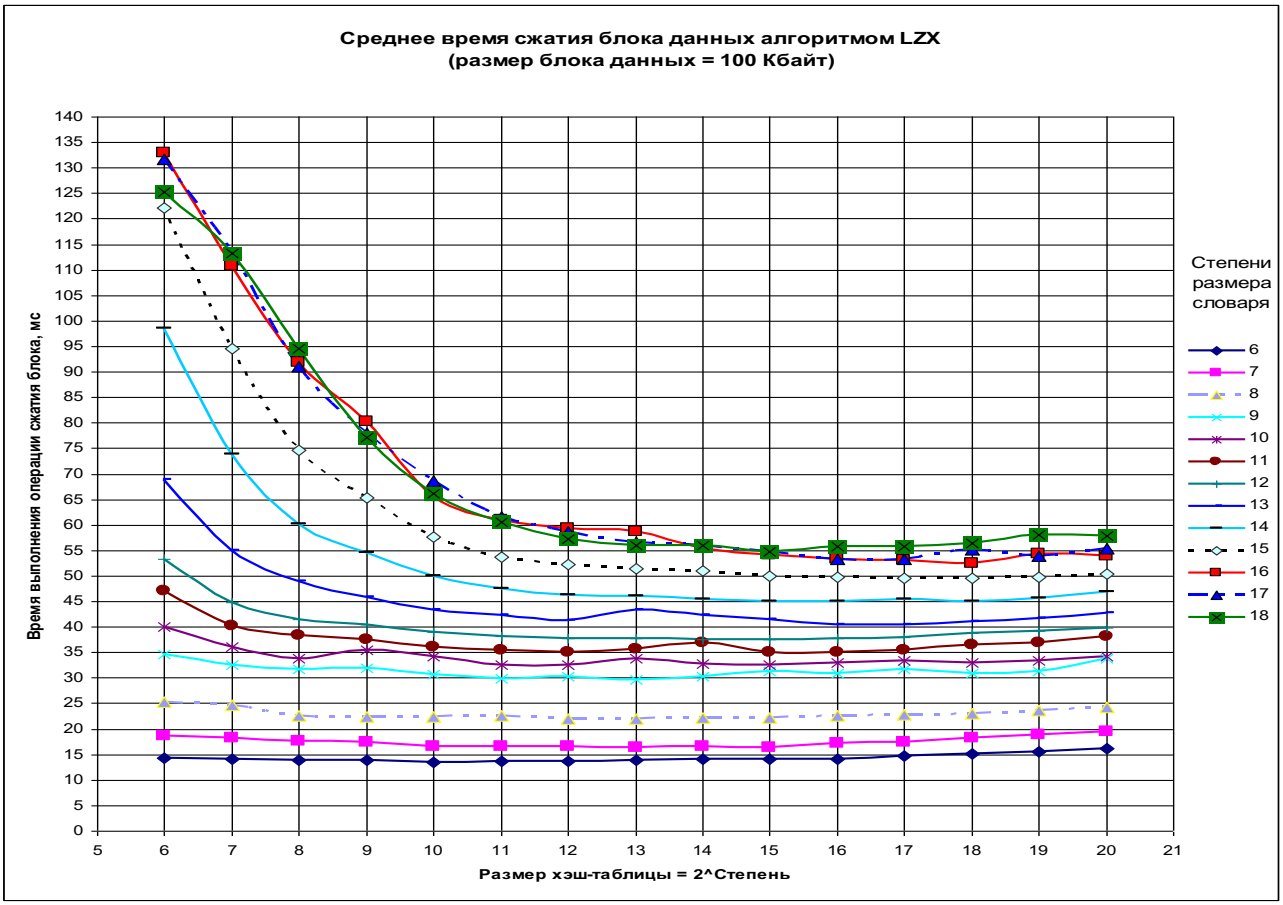
Суффиксальное дерево представляет собой структуру в виде дерева, в которой каждый лист является одним из суффиксов данного дерева, т.е. при движении из корня в любой из m листов проход осуществляется по n символам, которые представляют собой один из суффиксов строки. Более подробно о суффиксальных деревьях можно прочитать в [2]. Ниже приводится пример дерева для строки “XABXACBXA”.

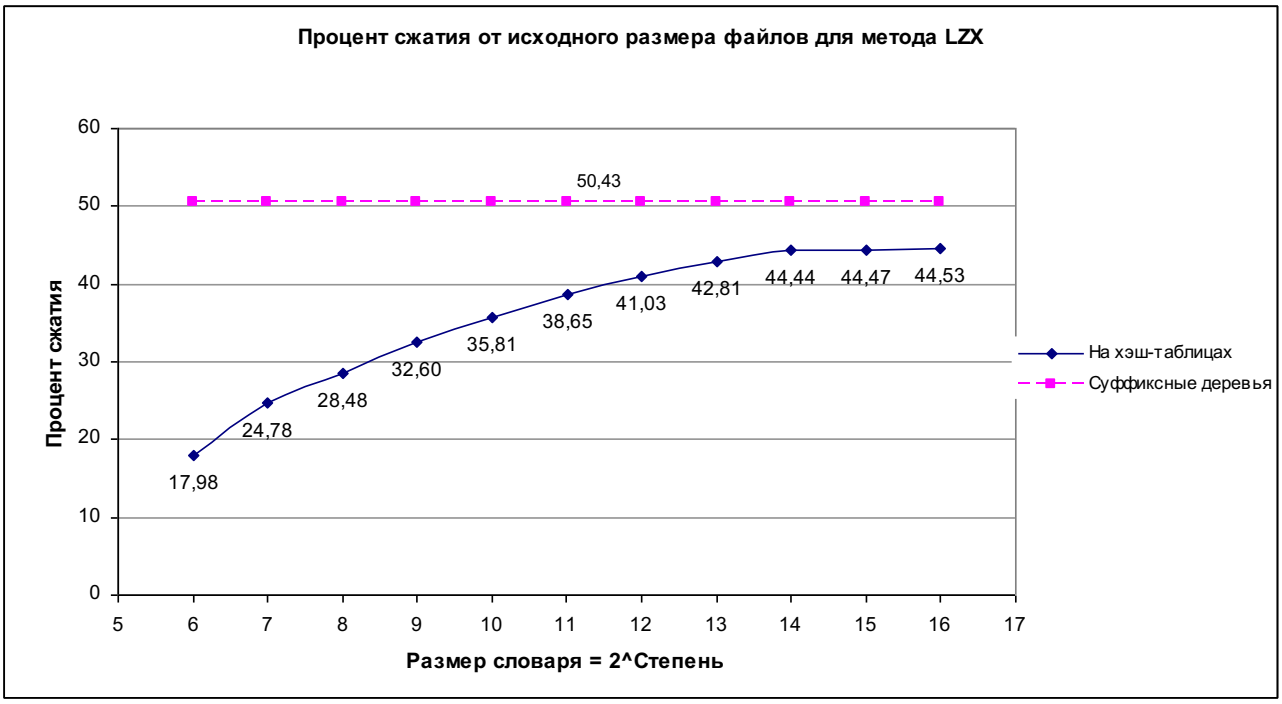
Как показывает рисунок 3, суффиксальное дерево отображает все суффиксы исследуемой строки и связи между ними. На рисунке изображено 9 суффиксов строки “XABXACBXA” и видно, что существует по крайней мере 5 связей между суффиксами строки. В дереве может возникнуть ситуация, когда один суффикс строки совпадает с префиксом другого суффикса, такое обстоятельство приводит к тому, что суффикс никак не завершается. Ситуация совпадения суффикса и префикса другого суффикса встречается в примере, который приведен на рисунке. Суффикса “XA” является префиксом суффикса “XACBXA” и префиксом суффикса “XABXACBXA”. Для решения данной ситуации предлагается каждый суффикс заканчивать специальным символом, который может встретиться только в конце суффикса и нигде более. Подробнее о работе с суффиксальными деревьями написано в [2], также в этом источнике приведен пример алгоритма сжатия LZ77 на суффиксальных деревьях.

Сравнение алгоритмов

Авторами была проведена работа по сравнению реализаций алгоритма сжатия LZ77, для чего была разработана программа ProtoArchive, которая имеет графический и консольный вид. При тестировании был выбран более предпочтительный консольный вид, т.к. он позволяет автоматизировать тестирование сразу для диапазона параметров определенного алгоритма. В программе используется собственная реализация алгоритма сжатия на хэш-таблице названная LZX и оформленная в виде отдельное библиотеки LZXLib. Для тестирования алгоритма на суффиксальных деревьях используется сторонняя реализация алгоритма на шаблонной библиотеке PATL (Practical Algorithm Template Library). Тестирование проводилось на стандартизованных пакетах для сравнения алгоритмов сжатия: CalgCC, ARTest, VYCCT, ACT. Более подробно о пакетах для тестирования алгоритмов сжатия можно прочитать в [1].

Ниже приведены графики тестирования на время выполнения операций сжатия и распаковки блоков данных с помощью алгоритма LZX:





Как видно из графиков, при увеличении размера хэш-таблицы, скорость падает. На больших размерах словаря скорость падения увеличивается. Рост времени сжатия блока после увеличения свыше 2^15 байт размера словаря прекращается. Процент сжатия снижается после 2^15 байт размера словаря. Скорость сжатия при увеличении размера хэш-таблицы свыше 2^14 не увеличивается.

Следовательно:

Оптимальный размер словаря 2^15 – 2^16 байт;

Оптимальный размер хэш-таблицы 2^14-2^17 байт;

При использовании большого словаря, требуется задавать положения индексов числами, максимальная величина которых равняется размеру словаря. Один из способов улучшить качество сжатия состоит в использовании кодов переменной длины.

Заключение

Использование суффиксных деревьев дает лучшие результаты по скорости и по проценту сжатия, чем реализация алгоритма LZ77 на хэш-таблицах. Алгоритм сжатия на суффиксных деревьях не позволяет изменение размеров словаря, в данной реализации используется понятие буфера, который должен быть не меньше 1 Мб. Поэтому для малого количества данных целесообразнее использовать реализацию на хэш-таблицах, для больших размеров сжимаемых данных стоит выбирать суффиксные деревья.

Литература

1. **Ватолин Д., Ратушняк А., Смирнов М., Юкин В.** ***Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео*****. - М.: ДИАЛОГ-МИФИ, 2002. - 384 с.**
2. ***Галфилд Д. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология. Издательства: Невский Диалект, БХВ-Петербург, 2003 г. 656 стр.***
3. ***Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн Алгоритмы. Построение и анализ. Издание 2-е Издательство: Вильямс, 2007 г. 1296 стр.***
4. **Дональд Кнут Искусство программирования, том 3. Сортировка и поиск = The Art of Computer Programming, vol.3. Sorting and Searching. — 2-е изд. — М.: «Вильямс», 2007. — 824 стр.**