УДК 681. 3.053

К.Ю. Балашов

О ДОСТИЖЕНИИ МАКСИМАЛЬНОЙ СТЕПЕНИ СЖАТИЯ НА ОСНОВЕ LZW-АЛГОРИТМА

Рассматриваются аспекты создания алгоритмов сжатия информации на основе LZW алгоритма с максимальной степенью сжатия

Введение

Данные, хранимые на различных носителях и передаваемые по каналам связи, обычно обладают значительной избыточностью. Использование алгоритмов сжатия информации позволяет повысить в несколько раз скорость обмена по каналам связи и во столько же раз экономить объем дискового пространства.

Как видно из рис. 1, на сегодняшний день существует множество подходов к сжатию информации и алгоритмов, их реализующих [1]. Наиболее распространенными на сегодняшний день являются словарные алгоритмы. Существуют две ветви словарных алгоритмов. Первая берёт начало от алгоритма LZ77 и основывается на замене строки символов в потоке на ссылку на ранее встретившуюся такую же строку и длину заменяемой строки. Вторая происходит от алгоритма LZ78. В алгоритмах этой ветви по ходу работы заполняется словарь. Если во входном потоке встретилась строка, уже присутствующая в словаре, она заменяется на соответствующий номер элемента словаря.

В настоящее время LZ77-подобные алгоритмы практически не используются самостоятельно - создано множество удачных комбинированных алгоритмов, где выходной поток LZ77-подобного алгоритма дожимается вероятностным контекстно-свободным алгоритмом [1]. Как правило, это (в порядке убывания быстродействия и повышения степени сжатия) алгоритмы Шеннона-Фоно, Хаффмана, арифметическое сжатие. Именно эти комбинированные алгоритмы успешно используются в большинстве современных универсальных архиваторов (ARJ, PKZIP, LHA, RAR).

Семейство LZ78 состоит из алгоритма LZW [2] и его незначительных модификаций. Собственно LZW-алгоритм практически не используется, так как его выходной поток содержит легко устранимую избыточность, вызванную чрезмерной разрядностью выходных кодов при частичном заполнении словаря. LZC - чисто практическая реализация LZW-алгоритма, применяемая в программе COMPRESS, используемой в системе UNIX и в графическом формате GIF [3].

Алгоритм LZT - разновидность LZW с удалением из словаря наименее используемых фраз.

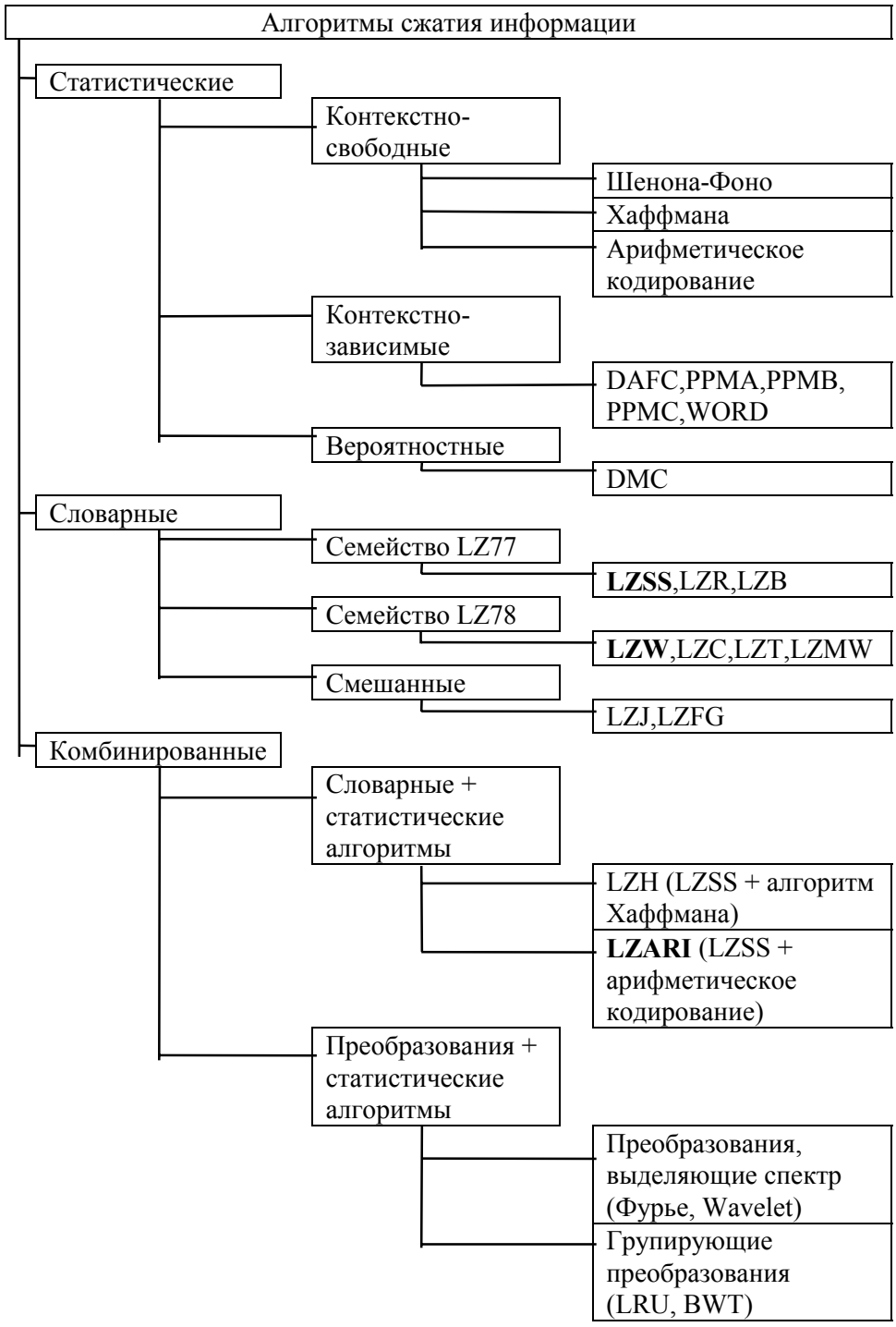

Рис.1. Алгоритмы сжатия информации.

В LZMW новые элемены словаря получаются конкатенацией двух существующих, а не элемента словаря и символа, как в LZW, что позволяет добиться более высокой скорости роста заменяемых строк и, в некоторых случаях, повышения степени сжатия [1].

Наиболее удачным алгоритмом, использующим замену строки на номер элемента словаря, является алгоритм LZFG [4], имеющий признаки как LZ77, так и LZ78-алгоритма. Его словарь построен на специальных деревьях, носящих название Patricia trees. Их главное отличие от деревьев словаря LZW-алгоритма - наличие дополнительных связей между деревьями.

В настоящей статье рассматриваются аспекты построения комбинированных алгоритмов на основе LZW и контекстно-свободных алгоритмов с целью получения алгоритмов с высокой степенью сжатия. В качестве контекстно-свободного алгоритма было использовано арифметическое кодирование, так как оно обладает наиболее высокой степенью сжатия в своей группе и в то же время достаточной гибкостью, что позволяет успешно использовать его в сложных комбинированных алгоритмах.

LZW-алгоритм

LZW-алгоритм разработан Terry A. Welch'ем на основе алгоритма LZ78 Лемпела и Зива. Он устраняет в потоке данных избыточность, вызванную появлением в различных местах потока одинаковых цепочек символов [1, 2, 3, 5].

Ядром LZW-алгоритма является словарь, содержащий все символы входного алфавита и некоторые строки из этих символов (рис. 2).

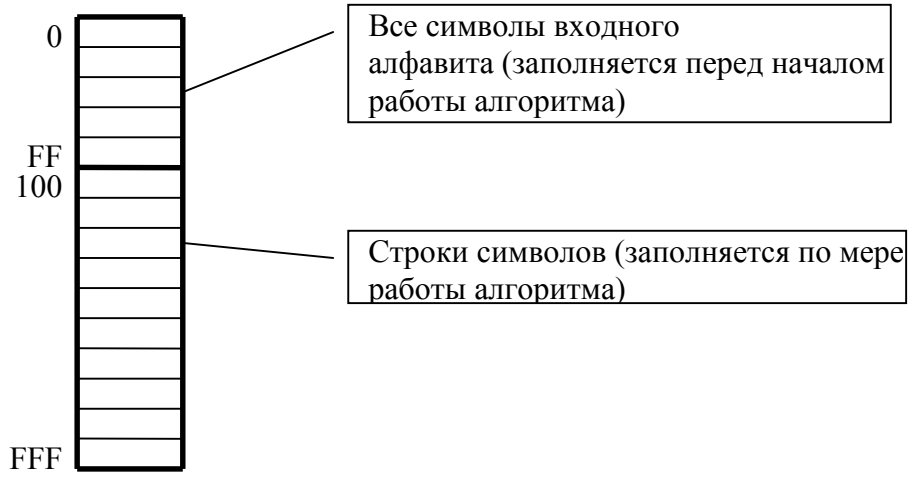

Рис. 2. Устройство словаря LZW-алгоритма. Номера ячеек приведены слева в шестнадцатеричном виде для 8-битных входных символов и словаря размером 4096 элементов

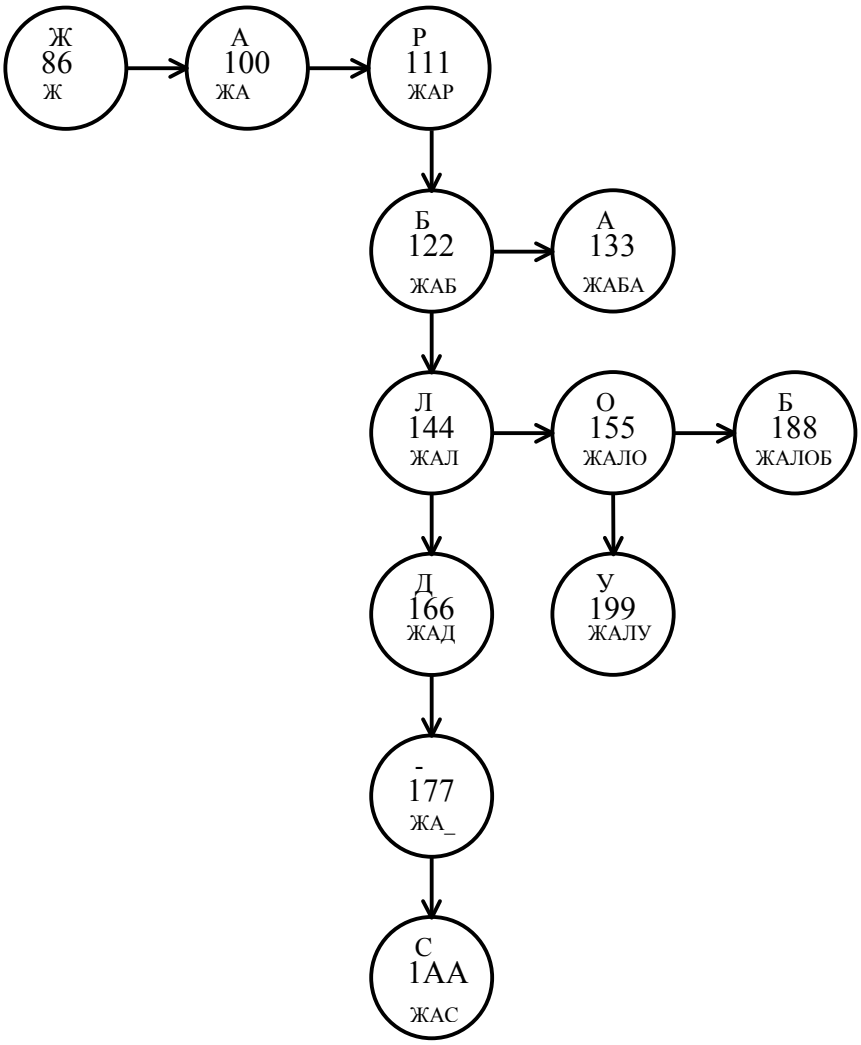

Рис. 3. Пример дерева LZW-алгоритма

До начала работы алгоритма в словарь входят элементы, в качестве строк содержащие каждый из символов входного алфавита. По мере работы алгоритма встречающиеся во входном потоке цепочки символов заносятся в словарь. При этом в выходной поток заносится номер элемента словаря, содержащий строку максимальной длины, совпадающую со строкой входного потока.

Словарь организован как совокупность двоичных деревьев и содержит по одному двоичному дереву на каждый символ входного алфавита. Каждый элемент словаря является в то же время вершиной одного из двоичных деревьев. Вершины деревьев являются элементами словаря, соответствующими символам входного алфавита. Каждая вершина соответствует некоторой строке, имеет одну входящую дугу и до двух выходящих. Первая из них указывает на вершину, которая соответствует строке, такой же, как данная, но на один символ длиннее

(связь <равно>), вторая - на вершину, соответствующую строке, имеющей такую же длину, как данная, но отличающуюся последним символом (связь <не равно>). В каждой вершине графа содержится также один символ.

Параллельно выборке символов из входного потока идет обход одного из деревьев. Алгоритм обхода следующий.
- выбирается первый символ из входного потока;
- выбирается соответствующее ему двоичное дерево в словаре;
- выбирается следующий символ из входного потока;
- далее для каждой вершины двоичного дерева
    если символ совпал с символом в вершине
        перейти к следующей вершине по связи <равно>;
        взять из входного потока следующий символ;
    иначе
        перейти к следующей вершине по связи <не равно>,
        продолжать с тем же символом;
- если ссылки дальше нет, поиск завершается, одна из пустующих вершин заполняется, добавляется соответствующая ссылка на нее из текущей вершины.

Пример дерева для буквы “Ж” представлен на рис. 3. Такое дерево сформировалось бы при поступлении на вход алгоритма следующей строки:
“...ЖА...ЖАР...ЖАБ...ЖАБА...ЖАЛ...ЖАЛО...ЖАД...ЖА_...ЖАС...ЖАЛОБ...ЖАЛОС...ЖАС...”

Арифметическое сжатие

Арифметическое сжатие предполагает знание вероятностей появления во входном потоке символов входного алфавита. Символы входного алфавита располагают в некотором фиксированном порядке. Промежуток от 0 до 1 делят между символами на начальные интервалы в соответствии с величинами их вероятностей. Оптимальная ширина начального интервала для символа с вероятностью появления во входном потоке p равна $-\log_2 p$. Работа алгоритма наглядно показана на рис. 4, где на четырех диаграммах схематически отображены первые 3 шага работы алгоритма для входного алфавита, состоящего из символов «A», «B» и «C» при поступении на вход алгоритма потока символов «B», «A», «B». Первая диаграмма показывает деление интервала (0,1) на начальные интервалы между символами входного алфавита до начала работы алгоритма. После поступления на вход алгоритма символа «B» рабочий интервал (X,Y) аналогичным образом делится между символами входного алфавита (диаграма 2) и т.д. Диаграмма 4 соответствует заштрихованному участку диаграммы 1.

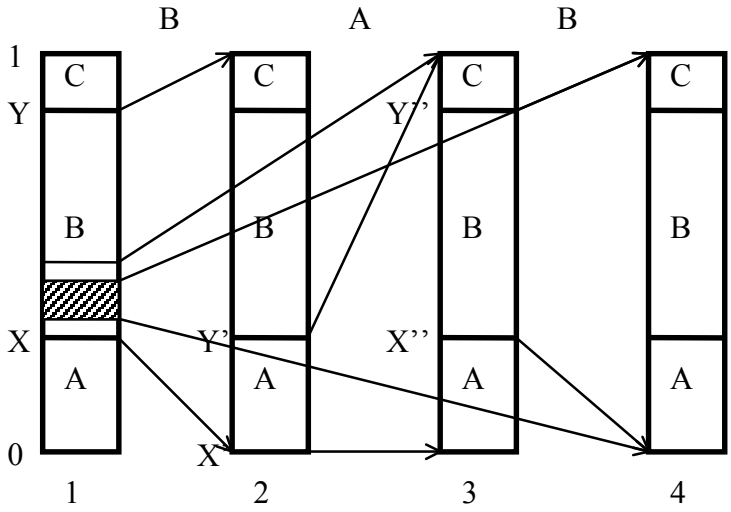

Рис. 4. Схема работы алгоритма арифметического сжатия

По мере обработки символов из входного потока рабочий интервал (X,Y), (X',Y'), (X'',Y''), ... сокращается. Насколько он сократится при обработке одного символа, зависит от ширины начального интервала, соответствующего символу.

Теоретически выходными данными алгоритма может служить любое число, входящее в рабочий интервал после обработки последнего символа входного потока. На практике же при каждом сокращении интервала в выходной поток помещаются все совпадающие старшие биты верхней и нижней границы рабочего интервала. Очевидно, что при поступлении символа с большей вероятностью появления и с более широким начальным интервалом рабочий интервал сократится меньше и меньше бит попадет в выходной поток. Таким образом, символы с большей вероятностью появления кодируются меньшим числом бит.

На практике чаще используется динамическая разновидность алгоритма, когда в начале его работы вероятности символов входного алфавита неизвестны и они вычисляются и динамически изменяются по мере работы алгоритма. Могут быть разные стратегии динамического вычисления вероятностей, характеризующиеся величиной наращивания вероятности встретившегося во входном потоке символа. Будем называть модели, в которых вероятность увеличивается сильно, моделями с высокой адаптивностью, а модели, в которых она увеличивается слабо - моделями с низкой адаптивностью. Очевидно, что для каждого конкретного случая существует своя оптимальная величина адаптивности.

LZW + арифметическое сжатие.

Аналогично тому, как в алгоритмах LZH и LZARI выходной поток алгоритма LZSS дожимается статистическими методами, можно

попытаться дожать выходной поток LZW-алгоритма арифметическим кодированием. Входной алфавит арифметического кодирования в этом случае будет переменным, его мощность будет колебаться от мощности входного алфавита LZW-алгоритма до размера словаря LZW-алгоритма.

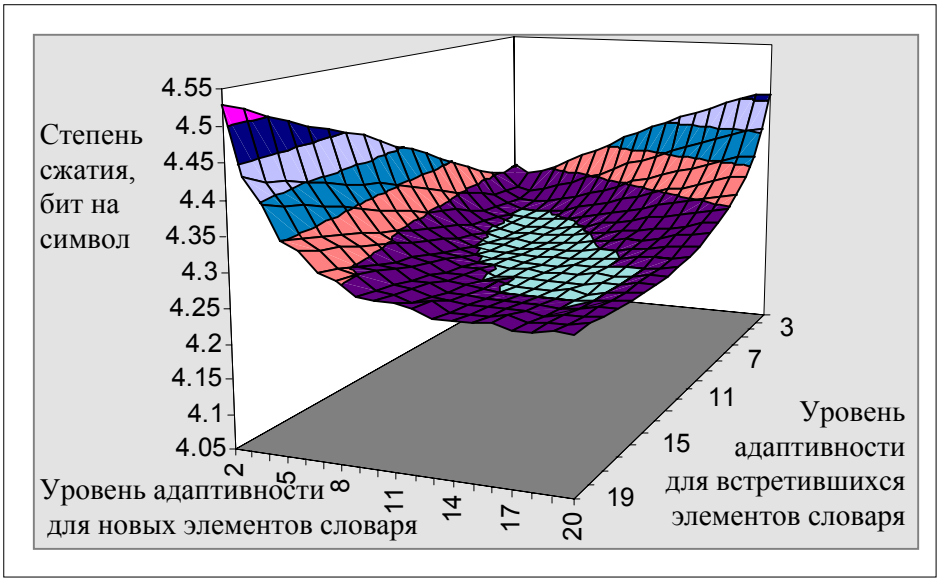

Рис.5. Зависимости степени сжатия от уровня адаптивности

Автором были проведены исследования с целью выяснить, как зависит степень сжатия такого составного алгоритма от изменения уровня адаптивности. В алгоритме можно менять как вероятность, присваиваемую новым элементам словаря, так и приращение вероятности кода, встретившегося в выходном потоке LZW-алгоритма. Результат получается примерно одинаковым для разных видов входных наборов данных и отличается только общим уровнем сжатия. Типичная диаграмма зависимости степени сжатия от уровня адаптивности приведена на рис. 5. В качестве набора входных данных взят художественный текст.

Дополнительное повышение степени сжатия

Во время проведения экспериментов автором был обнаружен новый, не известный ранее вид избыточности выходного потока LZW-алгоритма.

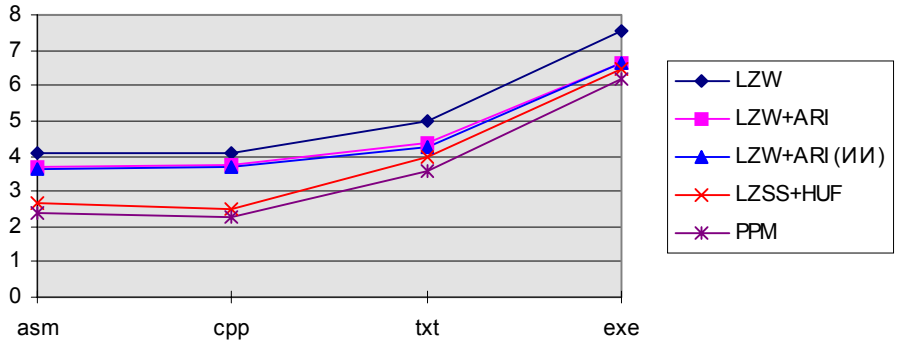

Рис.6. Сравнительная характеристика степени сжатия для разных алгоритмов на различных наборах данных

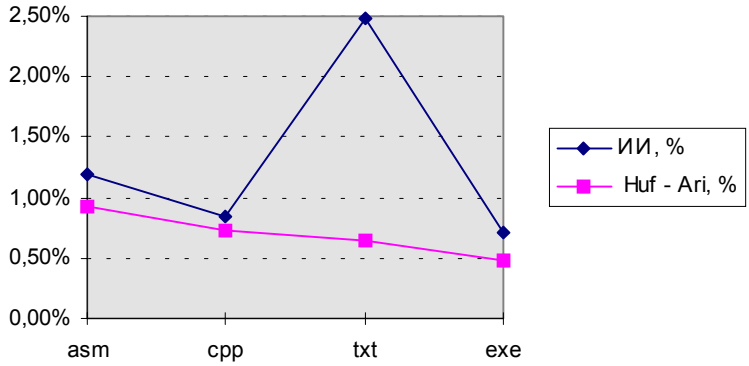

Рис. 7. Сравнительная характеристика выигрыша в степени сжатия при переходе с алгоритма Хаффмана на арифметическое кодирование и при устранении исключающей избыточности (ИИ)

Рассмотрим еще раз рис.3. Предположим, на выход алгоритма поступил код 100. Если бы во входном потоке следующим был символ «Р», то на выход поступил бы код 111, если «Б» - то 122 и т.д. Следовательно, следующий код не принадлежит к деревьям, начинающимся с этих символов. Все элементы словаря, принадлежащие к этим деревьям, можно исключить из рассмотрения при построении интервалов в арифметическом кодировании, что повысит вероятности (и размер начальных интервалов) других символов входного алфавита арифметического кодирования. Будем называть этот вид избыточности

исключающей избыточностью (ИИ). В результате получается выигрыш в степени сжатия порядка 1-2%.

Заключение

Как можно видеть из рис. 6, даже устранение исключающей избыточности не позволяет рассматриваемому составному алгоритму успешно конкурировать по степени сжатия с аналогичными алгоритмами на основе LZSS. Однако существуют два аспекта, вследствие которых обнаружение и устранение этого вида избыточности имеет важное значение. Во-первых, 1-2% для алгоритмов сжатия - это достаточно много. При дожатии выходного потока LZSS-алгоритма считается целесообразным переход от кодирования Хаффмана к арифметическому кодированию для повышения степени сжатия. Как можно видеть из рис. 7, где приведены сравнительные характеристики такого перехода и устранения исключающей избыточности для разных наборов входных данных, выигрыш от исключающей избыточности значительно больше.

Во-вторых, в данном исследовании использовалась наиболее примитивная модель распределения вероятностей выходного потока LZW-алгоритма. По-видимому, проведя соответствующие исследования, можно построить более совершенную и адекватную модель, что позволит повысить степень сжатия алгоритма и, возможно, довести ее до степени сжатия алгоритмов на основе LZSS. В этом случае устранение исключающей избыточности будет иметь первостепенное значение. То что выигрыш от ее устранения практически не зависит от уровня адаптивности алгоритма, позволяет надеяться, что на более совершенной модели она все также будет давать выигрыш в 1-2%.

Литература

1. Timothy Bell, Ian H. Witten, John G. Cleary. Modeling For Text Compression. ACM Computing Surveys. Vol.21, No.4 (Dec.1989), pp.557-591.

2.Terry A. Welch. A Technique for High-Peformance Data Compression. Computer, 17, № 6, 8-19 (1984)

3. Романов В.Ю. Популярные форматы файлов для хранения графических изображений на IBM PC.- М.:Унитех, 1992.

4. Fiala E.R. and Greene D.H. 1989. Data compression with finite windows. Commun.ACM 32,4(Apr.), 490-505.

5. Steve Apiki. Loss-Less Data Compression. Byte, March 1991.