Coordinate Scheme of Coding.

КСК относится к АС без потери качества и, как у других АС, у него возможны различные реализации, которые наиболее подходят в том или ином случае. Для того, чтобы понять принцип работы КСК полезно рассмотреть структуру архива, получающегося после работы алгоритма.

Структура архива.

$$A_1\left[fin\_bit, Ad_1^1, fin\_bit, Ad_2^1, fin\_bit, ..., Ad_{m-1}^1, fin\_bit\right], ..., A_n\left[fin\_bit, Ad_1^n, fin\_bit, ..., Ad_{k-1}^n, fin\_bit\right]$$

A_i - i-ая буква алфавита.

Ad_k^i - адрес k+1 – го экземпляра i – ой буквы (это не опечатка: именно к+1 – го, а не к –го – почему: будет разъяснено позже).

fin_bit - этот бит нужен для того, чтобы распаковщик знал все ли экземпляры буквы А записаны в файл.

Принцип работы распаковщика.

Все буквы в архиве записаны в том порядке, в котором первые вхождения этих букв встречаются в исходном файле (распакованном файле), если его просматривать с начала до конца: первый экземпляр каждой буквы записывается по ближайшему от начала файла не занятому адресу.

Условия получения положительного результата.

Допустим: у нас файл размером 4Гб и размер каждой буквы 4 байта – для прямой адресации нам нужно $\log_2 \frac{длина\_файла}{длина\_буквы}$=30бит.

Выведем формулу для подсчёта сколько бит тратится на букву с n вхождениями в архиве: (кол-во fin_bit=n)+(кол-во адресов=n-1)*(длина поля адреса=30)+(длина буквы=32) = $= n + 30(n-1) + 32$. Из формулы видно, что на букве с одним вхождением: идёт потеря одного бита; с двумя: результат нейтральный; начиная с трёх: получаем эффект сжатия. Кстати, заметим, если бы мы писали б в архиве адреса первых вхождений – для положительного эффекта, при данных условиях, потребовалось бы не менее 33 вхождений буквы.

Dynamic Scheme of Coordinate Coding.

DSCC является расширенной версией CSC, в которой содержатся две нижеследующие методики или какая – либо из них.

<u>Сужение поля адресации.</u>

При заполнении файла на процент равный $\sum_i \frac{1}{2^i}, i = 0, ..., n$ можно сократить длину поля на 1 бит: i=0 – длина_поля=длина_поля; (50%)i=1 – длина_поля --;(75%) i=2 – длина_поля -- и.т.д. Возникает закономерный вопрос как(??!!) – как при сужении обеспечить адресацию, ведь объём

файла прежний. Использование битовой карты позволяет сводить адресацию из общего диапазона к диапазону свободных адресов.

Привязка вторичных кодов.

По мере заполнения файла, образуется перечень использованных кодов, каждый из которых больше встречаться не будет – все эти коды заносятся в массив. Итый элемент массива привязывается к i - му свободному адресу в файле. Таким образом можно получить доп. процент сжатия за счёт экономии на некотором количестве финальных бит. Ну, и ещё, что стоит отметить – массив должен переопределяться заново на каждом шаге сужения, впрочем, это очевидно.

Dual Scheme of Coding.

Все буквы разбиваются на две категории: одна кодируется координатной схемой, а другая словарной. При распаковке сначала декодируется координатка, - затем все свободные ячейки заполняются из второй части архива (словарной).

Segment models.

Прочитав всё вышеизложенное, можно заметить главную проблему методик: ресурсоёмкость, к тому же размер обрабатываемого файла зависит от длины буквы.
Для того, чтобы преодолеть весь этот негатив, имеет смысл, проводить обработку файла сегментами (адресация в сегменте, а не по всему файлу). Это позволит:
A. Работать с меньшей длиной буквы.
B. Терять на бесперспективных сегментах 1 бит.
C. Возможна обработка файла любой длины.
D. Использовать DSCC в полной мере: при адресации по файлу это практически не возможно.

Структура архива.

$$[archive\_bit, Seg1],[archive\_bit, Seg2],.....,[archive\_bit, SegN].$$

Устройство сегмента соответствует либо CSC, либо DSCC.

Ещё стоит отметить, что, если файл разбивается на равные сегменты, то указывать длину каждого сегмента в архиве не требуется.

ABC System With Limit of Losses.

Все буквы разбиваются на две категории: в первой каждая буква получает уникальный код – во второй все имеют один и тот же код минимальной длины. В архиве буквы второй категории должны располагаться в той же очередности, что и в исходном файле.

Заметки по различным схемам и их реализациям.

Некоторые пояснения.

Схема A – самая примитивная скачать исходник можно отсюда

Схема Б(ограниченная реализация сегментной модели) – cpp файл (хедеры и либу брать из сырца схемы А) бинарник
Схема В (полный DSCC – ещё не реализован).
Схема Г (Ещё не реализована, её описание далее по тексту).

Замечания по схеме "Б".

Данный вариант является сегментной моделью с длинной сегмента в 65 байт и длинной буквы в 8 бит. Тесты показали, что данная реализация уступает программам 7zip и Rar на различных файлах, в том числе малоизбыточных файлах. Для улучшения существующей ситуации можно внедрить в схему исключение "плохих" (несжимаемых) сегментов, тоесть они (сегменты) будут помечаться в архиве как несжимаемые, точней говоря, для каждого сегмента в архиве будет бит, сигнализирующий декодеру о том, как обрабатывать текущий сегмент. Подобная доработка несколько увеличит накладные расходы на код, зато выгода очевидна: верхняя граница потерь без учёта несжимаемых сегментов составляет 12.5% , а потолок выигрыша равен 4, 167% - при учёте "плохих" сегментов максимальный предел потерь будет состовлять

$$\frac{1}{Длина\_сегмента\_в\_байтах * Длина\_буквы\_в\_битах} = \frac{1}{65*8} = 0{,}0019231$$

.

Для примерной оценки эффективности схемы, с учётом "нехороших" сегментов, в текущей внедрена возможность подсчёта, сколько байт набежало из – за финальных бит (на консоль выводится параметр wrong). Зная эту информацию, можно примерно подсчитать возможную степень сжатия по следующей формуле: $Возможный\_размер\_архива < размера\_архива\_по\_схеме\_Б - WRONG + 0{,}0019231$.

Плюсом реализации схемы "Б" является применение индексной сортировки, что позволяет отвечать на вопрос "создан ли для текущей буквы пакет" за одну итерацию. Правда, на больших сегментах придётся использовать менее быстрые бинарные сортировки, так как ресурсоёмкость индексной сортировки по оперативной памяти состовляет:

$$O\left(Длина\_буквы\_в\_битах * 2^{Длина\_буквы\_в\_битах}\right)$$

В коде присутствует фактор, сильно замедляющий скорость работы программы – использование операции сдвига в массиве при считывание следующей(его) буквы/адреса: эту операцию можно заменить операцией побитного чтения в массиве.
Для быстрого теста сегмента по вопросу его сжимаемости можно использовать нижеследующий код:

```
int test_good_or_bad_segment(char &, int end_of_buf/*сколько элементов в массиве*/)
        {
        bool flag[256]; // флаги: встречалась ли текущая буква раньше
        int count_abc=0; // сколько всего букв в алфавите;
        int i=0; // счётчик
        ZeroMemory(flag, 1, 256); // устанавливаем флаги в состояние ложь
        while(i<end_of_buf) // посредством индексной сортировки считаем размер алфавита
                {
                if(flag[buff[i]]) count_abc++;
```

```
        else flag[buff[i]]=true;
        i++;
        }
if(floor((double)65/count_abc)>=3) return 1; // сегмент “хороший”
else return 0; // no
}
```

Замечания по схеме “В”.

Следует отметить ряд вещей: в отличие от схемы “Б”, ДКСК требует использование битовой карты как в упаковщике, так и в распаковщике – на ней завязано сразу два вопроса: сужения поля адресации и привязка вторичных кодов. На малых сегментах имеет смысл использовать индексную сортировку при формировании пакетов для конкретной буквы. Для выяснения вопроса имеет ли свободная ячейка привязанный код, к сожалению, построить более эффективный алгоритм, чем последовательный перебор битов карты врят ли получиться. Алгоритм определения того, что сегмент “хороший” тоже неоднозначен: с одной стороны: можно использовать вышеприведённый пример для КСК(с поправкой, что ДКСК может давать эффект сжатия уже при среднем вхождении каждой буквы в сегмент 2 раза) – с другой стороны оценка 2 не совсем верна: она может быть меньше, например 1, 97; к тому же на неё влияет и распределение букв в сегменте, так как этот фактор важен для вторичных кодов. Таким образом, наиболее эффективным способом определения перспективности попытаться его сжать, что плохо для скорости программы.

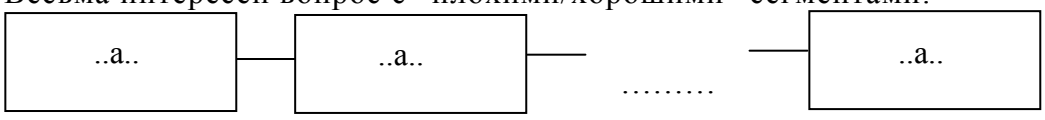
Координатная схема кодирования с относительной - опорной адресацией и фиксированной длиной прыжка (Схема “Г”).

В этой схеме адрес текущего вхождения буквы равен адресу предыдущего её вхождения плюс смещение относительно этого адреса. Понятие “опорная” заключается в том, что смещения первой буквы считаются без учёта смещений других букв; второй – с учётом смещений первой буквы; третей – с учётом первой и второй букв и. т. д. Если длина прыжка меньше расстояния между двумя вхождениями буквы, тогда устанавливается финальный бит в ложь. Непрерывная относительная адресация не имеет смысла: ввиду неравнамерного распределения букв в файле выигрыш в одной части файла будет съедатся в других, да, и ещё, что стоит отметить, что придётся выделить код, который будет сообщать декодеру, что нужен ещё один прыжок, - а это не лучшим образом будет влиять на эффективность.

Очерёдность опорных букв не может быть произвольной она должна быть строго последовательной иначе распаковка не будет однозначной (нельзя вычислять смещения первой буквы с учётом итой буквы).

Очерёдность может быть статичной, а может переопределяться динамично на основе модели предсказаний, впрочем, этот подход на малоизбыточных данных неэффективен: модель требует начального состояния (дополнительные накладные расходы на код); прирост выигрыша, в основном, незначителен, зато ресурсоёмкость реализации резко возрастает.

Весьма интересен вопрос с “плохими/хорошими” сегментами:



как видно из вышеприведенного примера, соседство “плохих” сегментов может приводить к образованию выигрышных цепочек букв; задача анализа осложняется тем, что каждая буква, как правило, имеет выигрышную цепочку своей длины, так

что строить алгоритм в общем виде не имеет смысла – он будет слишком медленным. Можно применить частное решение этой проблемы: тестировать не конкретный сегмент, а их цепь (например, цепь из 7 сегментов) – в итоге получаем два положительных факта: цепь может быть протестирована функцией test_good_or_bad_segment; накладные расходы на код, по сравнению со схемой “Б”, снижены.

Сравнение схем “В” и “Г”.

Допустим, имеется цепочка из трёх сегментов, у которых среднее количество вхождений каждой буквы составляет 1,07 на сегмент, а, в расчёте на цепочку, 3,02: схема “В” неспособна получить положительный результат в данном примере; схема “Г” имеет вероятность положительного исхода.

Ремарка.

Координатные схемы чрезвычайно гибки: могут быть адаптированы как под текстовые файлы, так и под различные бинарные файлы с заведомо малой избыточностью. Мной разрабатываются, на основе расширенного определения избыточности, более совершенные алгоритмы сжатия.

Licence:

Date: **30.05.2007**

WWW: http://xproject-all.narod.ru/prgsale.htm