A. Gelbukh. *Minimization of the number of disk accesses in dictionary-driven morphological analysis* (in Russian). J. Nauchno-Tehnicheskaya Informaciya (NTI), ISSN 0548-0027, ser. 2, vol. 6, Moscow, Russia, 1991, pp. 15–21.

МИНИМИЗАЦИЯ КОЛИЧЕСТВА ОБРАЩЕНИЙ К ДИСКОВОЙ ПАМЯТИ ПРИ СЛОВАРНОМ МОРФОЛОГИЧЕСКОМ АНАЛИЗЕ

А.Ф.Гельбух

В рамках задачи морфологического анализа словоформ флективного языка рассматривается задача считывания из словаря основ на диске информации об основах слова для всех возможных вариантов морфологического разбора символьной цепочки. Задача сводится к нахождению в БД всех записей, ключи которых являются начальными подцепочками предъявленной цепочки. Минимизируется количество элементарных обращений к диску при жестких ограничениях на занимаемую оперативную память. Предлагается структура словаря, при которой получение всей необходимой информации гарантируется с первого же обращения; приводятся алгоритмы упаковки и распаковки словаря и поиска в нем.

ВВЕДЕНИЕ

Широко известная задача морфологического анализа словоформ флективного языка ставится следующим образом. Берется буквенная цепочка (словоформа), являющаяся формой некоторого словарного слова (лексемы). Путем расчленения словоформы на основу и один или несколько суффиксов (окончаний) необходимо определить, к какой конкретно лексеме относится данная форма и какие грамматические категории (т.е. род, падеж, число, лицо, время и т.п.) выражает [1]. Например, цепочка «*стекло*» расчленяется на *стекл-о* — СТЕКЛО, именительный падеж, или *стек-л-о* — СТЕЧЬ, прошедшее время, средний род.

Чтобы определить соответствующую лексему, необходимо выделить из данной словоформы все возможные варианты основы, для чего обратиться с ней к словарю основ. В статье словаря хранятся сведения, позволяющие проверить сочетаемость конкретной основы с данными суффиксами. Многие прикладные системы опираются на крупные словари (100 тыс. основ и более), размещенные на диске [2, 3]. В этих условиях очень важно предельно сократить число обращений к диску.

В данной работе рассматривается такое размещение основ по блокам постоянной длины, составляющим крупно словарь основ, при котором число обращений к словарю сокращается до предела. На наши результаты не влияет ни деление словаря на части по соображениям частоты употребления отдельных слов, ни кэш-буферизация или выделение оперативной части словаря.

Рассматривается строго флективная модель языка [4], но аффиксы предполагаются присоединенными к основе только справа, префиксы же либо рассматриваются как часть основы, либо считаются отделенными от буквенной цепочки до обращения к процедуре поиска в словаре. В случае супплетивизма основ (*человек-∅*, но *люд-и*; *ид-ти*, но *ше-л*) каждая из основ лексемы считается записанной в словарь независимо, а ссылки между ними, если это необходимо, обеспечиваются отдельным механизмом, здесь не рассматриваемым. Случаи чередования букв в основе (*пиш-у*, но *писа-л*; *станок-∅*, но *станк-а*) формально

сводятся к супплетивизму основ. Таким образом, считывание из словаря всех основ анализируемой словоформы является считыванием тех основ из хранящихся в словаре, которые являются начальными (левыми) частями анализируемой буквенной цепочки и удовлетворяют условиям грамматических согласований с оставшейся правой частью.

В данной работе рассматривается более общая задача, имеющая и самостоятельную ценность: получить из словаря все хранящиеся в нем основы, являющиеся начальными подцепочками анализируемой символьной цепочки, затратив на это минимум элементарных обращений к диску. Показывается, что все такие основы могут быть получены с первого же обращения к диску.

Такая общая постановка задачи обеспечивает дополнительные преимущества. Для поиска основ не требуется предварительно выделять искомое слово из входного потока символов, определяя его конец в потоке. Это, в свою очередь, позволяет записывать в словарь основы вроде *т.е.*, *во что бы то ни стало*, *PC/XT/AT* или *вице-президент* [3]. Более того, работа процедуры обращения к словарю становится полностью не зависимой от процедур анализа грамматической информации и не требует никаких лингвистических знаний. Появляется возможность определять основу при неизвестном окончании, что, в частности, делает простым и естественным алгоритм обработки сложных слов (путем последовательного вычленения основ слева направо). Последняя возможность может оказаться очень полезной при обработке текстов по химии или для анализа текстов на немецком языке, а также при исправлении ошибок в окончаниях.

СТРУКТУРА СЛОВАРЯ

Переформулируем нашу задачу в терминах СУБД [5]. Словарь рассматривается как простейшая база данных, отдельными записями которой являются словарные статьи, соответствующие основам, а их ключами — сами основы. Подсистема доступа к БД исполняет такие функции СУБД, как занесение информации и ее выборку по заданному условию на ключи выбираемых записей.

Задача выборки формулируется так: выбрать из базы все записи, ключи которых являются начальными подцепочками предъявленной буквенной цепочки.

Длина (конец) цепочки не задается при обращении к СУБД, поскольку результат не зависит от нее (см. также последний абзац Введения). Цепочка, начиная с ее первой буквы, рассматривается как неограниченная, однако ее всегда можно ограничить справа любым символом, отсутствующим в алфавите ключей. Можно также считать ее ограниченной максимальной длиной ключа записи в БД. Известные автору существующие СУБД не ориентированы на решение такой задачи.

Будем считать, что соображения экономии оперативной памяти не позволяют считывать в память сразу значительный объем входного текста и/или словаря. В процессе работы программы анализа входной буквенный поток должен считываться ею последовательно, а результаты анализа должны быть доступны сразу после считывания очередного полного слова. Выделение слова из входного потока (ср. *во что бы то ни стало*, *т.е.*) также входит в задачу. Минимизации подлежит количество элементарных чтений блока диска после предъявления процедуре

обращения к словарю очередной буквенной цепочки до получения всех записей из словаря, ключи которых вкладываются в нее слева (и установления того, что других таких записей в БД нет).

Уточним понятие элементарного обращения к диску. Из-за фрагментированности файлов в современных операционных системах и из-за аппаратной организации типового жесткого диска следует считать [6], что элементарной операцией является считывание строго определенного небольшого блока, например, сектора (обычно 512 байт) или кластера (обычно порядка двух Кбайт). Считывание более чем одного кластера часто требует перепозиционирования головки чтения, и время считывания в этом случае складывается из времени считывания отдельных кластеров.

Рассмотрим словарь, упорядоченный лексикографически (по алфавиту). Лексикографический порядок, в частности, позволяет применить известный способ сжатия текста по Куперу [5], сводящийся к тому, что вместо повторения в очередной строке тех ее первых букв, которые совпадают с первыми буквами предыдущей строки, указывается только количество совпавших букв [7].

Зафиксируем некоторую величину блока. Добавив в текст словаря, где это требуется, пробелы, добьемся того, чтобы ни одна его запись не пересекала границу блоков. Это всегда возможно, если в нем нет слишком длинных записей. Относительное увеличение объема словаря при этой операции не превышает среднего отношения размера записи к размеру блока, что в нашей реализации составило около 3%. Сжатие по Куперу применяется только внутри каждого блока и не затрагивает его начальную запись.

В памяти размещается индексный массив, содержащий для каждого блока ключ его первой записи. Заметим, что лексикографический порядок позволяет в таком массиве хранить каждый ключ не целиком, а только до той позиции (включительно), по которой он отличается от ключа последней записи предыдущего блока; этого достаточно для четкого различения блоков при поиске. Сам индексный массив также можно разбить на части, сжать их по Куперу и организовать доступ к ним через аналогичный индексный массив.

Казалось бы, такая структура, аналогичная известным B*-деревьям [8], непосредственно применима для наших целей: при поиске можно найти алфавитное место предъявленной цепочки в индексном массиве (алфавитным местом мы называем лексикографически максимальную из имеющихся в словаре записей, не большую предъявленной; если словарь содержит несколько таких записей с одинаковыми ключами, то следует рассматривать последнюю из них по расположению в словаре), вызвать с диска соответствующий блок и найти алфавитное место той же цепочки в нем. Затем следует обрезать цепочку по первой же букве, по которой она не совпадает с найденной в словаре, и повторить процесс для такой укороченной цепочки (прием сообщен автору Г.Б.Савиной).

Рассмотрим, например, поиск для цепочки «*парах*, *которые*...» (лексема ПАР либо ПАРА; см. выше замечание о том, что буквенный поток рассматривается без ограничения справа). Пусть ее алфавитное место в словаре расположено за ключом *парафин-*. Обрезаем цепочку по позиции несовпадающих букв «*ф*»/«*х*»: «*пара*» и повторяем поиск. Пусть теперь ее алфавитное место найдено после ключа *пара-*

(приставка ПАРА- в словах типа парапсихология). Поскольку несовпадающих букв нет, обрезаем только одну букву справа: «*пар*» и находим искомую основу. Продолжение поиска даст основу *па-* глагола ПАСТЬ (*па-л*) и существительного ПАЙ.

Однако здесь возникает следующая проблема: записи с ключами *парафин-*, *пара-*, *пар-* и *па-* не обязаны находиться в одном блоке словаря. Еще хуже ситуация будет для слова «*пары*», алфавитное место которого будет после всех слов *пароход*, *паровоз*, ..., а истинная основа *пар-* в имеющемся у автора словаре отстоит на десять 512-байтных блоков от него.

Предлагается следующее решение: продублировать в каждый блок словаря все те записи из других блоков, при поиске которых на первой итерации возможно обращение к данному блоку. Рассмотрим, насколько это увеличит объем словаря, и покажем, что увеличение будет незначительным.

Заметим, что записи, которые могут понадобиться на второй и дальнейших итерациях поиска, как видно из приведенного выше примера, имеют ключи, вкладывающиеся слева в ключ, найденный при первой итерации (в примере это ключ *парафин-*, а искомыми были записи с ключами *пара-*, *пар-* и *па-*). Таким образом, в каждый блок достаточно продублировать те записи, ключи которых вкладываются слева в ключ хотя бы одной записи из данного блока.

Покажем, что на самом деле дублировать придется лишь записи с ключами, вкладывающимися в ключ только первой записи блока, а все остальные уже есть в блоке (напомним, что словарь, не считая «лишних» продублированных записей, упорядочен лексикографически). Для этого покажем, что если некоторый ключ вкладывается слева в ключ из данного блока, то он либо вкладывается в первый ключ блока, либо лексикографически больше него. В последнем случае запись, имеющая такой ключ, уже расположена в данном блоке и не требует дублирования. Действительно, ее ключ больше ключа первой записи блока. Но ее ключ вкладывается слева в ключ некоторой записи из данного блока, и, по закону лексикографического порядка, лексикографически меньше него. Следовательно, запись расположена между двумя записями одного блока и сама расположена в нем.

Для доказательства предположим, что ключ K вкладывается в ключ B некоторой записи, входящей в рассматриваемый блок. Пусть B состоит из букв b_1, b_2, ..., b_n, K - из букв k_1, k_2, ..., k_l, $l < n$, а ключ A первой записи блока -–из букв a_1, a_2, ..., a_m,. Поскольку K вкладывается слева в B, то K состоит из тех же букв: $k_1 = b_1$, ..., $k_l = b_l$. Пусть K не вкладывается слева в A, т.е. буква k_i ключа K не совпадает с буквой a_i ключа A. Тогда либо $a_i < k$, либо $a_i > k_i$. Будем считать, что i — минимальный номер несовпадающих позиций в A и K, и, следовательно, в A и B. Поскольку A лексикографически меньше B, то $a_i < b_i$. Возвращаясь к тому, что $k_i = b_i$ — первая слева буква в K, отличающаяся от соответствующей буквы в A, получаем, что K лексикографически больше A. Данный вывод получен в предположении, что K не вкладывается слева в A.

Рассмотрим, например, словарь, содержащий ключи *па-*(*л*), *пад-*(*у*), *пар-*(*а*), *параграф-*(∅), *паров-*(*ой*), *паровоз-*(∅), *паровозн-*(*ый*), *пароход-*(∅). Пусть некоторый блок содержит записи, начиная с *параграф-* и кончая *пароход-*. Тогда вложенными, например, в ключ *паровозн-* являются ключи *па-*, *пар-*, *паров-*,

паровоз-, однако только *па-* и *пар-* подлежат дублированию в данный блок из предыдущих; те же, что включают позицию первого несовпадения «*а*»/«*о*» ключей *параграф-* и *паровозн-*, уже находятся в рассматриваемом блоке.

Поскольку в приведенном выше рассуждении в качестве ключа *B* можно полагать любой ключ из блока, окончательно получаем: в каждый блок следует продублировать только те записи, ключи которых вкладываются слева в ключ первой записи данного блока. Доказательство окончено.

Более того, если про некоторые записи дополнительно известно, что их ключи вообще не могут присоединять непустые окончания либо присоединяют только окончания, лексикографически меньшие соответствующего остатка первого ключа блока, то такие записи можно в данный блок не дублировать. Так можно избежать дублирования словарных статей предлогов типа *о*, *по* и т.п. В случае же непустых окончаний некоторую дополнительную выгоду из последнего замечания можно извлечь, изменив, если это допустимо, алфавитный порядок букв так, чтобы первые буквы наиболее часто (по словарю) используемых окончаний оказались в начале алфавита. Под первой записью блока всюду далее мы будем понимать первую без учета продублированных.

В имеющемся у автора словаре 512-байтный блок содержал после сжатия по Куперу в среднем около 30 записей, а продублировано в него было в среднем одна — три, редко четыре — пять записей (без применения обсуждавшегося в предыдущем абзаце приема). Объем словаря увеличился менее чем на 10%, что можно считать вполне приемлемым.

Заметим в заключение, что при расположении продублированных записей в начале блока алгоритм поиска в одном блоке не претерпит никаких изменений, т.к. блок по-прежнему остается лексикографически упорядоченным, и в нем обычным образом могут быть найдены искомые записи, но только в этом случае гарантируется, что они в нем найдены все и с первого обращения к дисковой памяти.

АЛГОРИТМЫ ПОИСКА И ФОРМИРОВАНИЯ СЛОВАРЯ

Рассмотрим алгоритм формирования словаря со структурой, обсуждавшейся в предыдушем разделе. Алгоритм осуществляет импорт БД (словаря) из файла, записи которого (словарные статьи) уже упорядочены лексикографически по ключам (основам). Записи входного файла поступают на вход алгоритма последовательно по одной, выходной массив внутреннего представления БД также формируется последовательно по мере поступления очередной входной записи.

Предварительно рассмотрим конструкцию, которую мы назовем стеком вложенных ключей. Это стек (организованный по принципу «последним пришел — первым ушел»), элементами которого являются записи БД — словарные статьи словаря. Стек вложенных ключей обновляется при поступлении каждой новой записи из входного файла.

В начале работы стек пуст. При поступлении очередной записи из входного файла с вершины стека последовательно удаляются (и теряются) все элементы, ключи которых не вкладываются слева в ключ поступившей со входа записи.

Процесс удаления останавливается, когда стек исчерпан либо на вершине стека встречена запись, вкладывающаяся слева в прочтенную. Затем новая запись помещается на вершину стека, и процесс повторяется: из входного файла вновь считывается запись, лишние удаляются из стека, а новая помещается в него, и т.д.

Как видно из алгоритма работы с таким стеком, гарантировано, что ключи всех находящихся в стеке записей вкладываются слева друг в друга. Более того, поскольку входной файл упорядочен лексикографически, то по завершении операций, связанных с внесением в стек очередной прочтенной со входа записи, в стеке находятся все записи словаря (включая ее саму), ключи которых вкладываются в ключ новой записи. Действительно, все они расположены выше по файлу, чем текущая запись, и ранее были внесены в стек. Но они не были удалены из стека. Доказательство этого факта аналогично приведенному в предыдущем разделе рассуждению с ключами *K*, *A* и *B*, и мы не будем его приводить.

Перейдем теперь к рассмотрению алгоритма формирования внутреннего представления БД. Нам необходимо расположить прочтенные со входа записи в блоках, продублировав в каждый блок все записи, ключи которых вкладываются слева в ключ первой записи блока. Лексикографическое упорядочение будет получено автоматически, поскольку так упорядочен входной файл.

Читая запись за записью входной файл, будем (не забывая произвести, если это нужно, куперовское сокращение) копировать их в выходной файл и попутно обновлять стек вложенных ключей так, как это обсуждалось выше. Однако, если длина очередной предназначенной к выводу записи такова, что выводимая запись должна пересечь границу блоков (в каждый момент известно, на каком байте выходного массива находится указатель записи), вместо ее вывода будут предприниматься следующие действия.

Вывод очередного блока завершается, выходной массив дополняется пробелами до границы блока, начинается вывод нового блока. В заголовок блока можно поместить уменьшенное на единицу число записей в стеке вложенных ключей (это может понадобиться для распаковки (экспорта) внутреннего представления БД в текстовый файл, а также для операций пополнения словаря). Куперовский процесс прерывается, т. е. первая запись блока выводится не сокращенной. На выход копируется все содержимое стека вложенных ключей, начиная с самой глубокой записи и кончая вершиной. Заметим, что на вершине стека расположена новая, не уместившаяся в предыдущем блоке запись (именно поэтому выше мы из числа элементов вычли единицу), а остальные его элементы суть продублированные в данный блок записи, содержащиеся в предыдущих блоках, ключи которых вкладываются слева в ключ первой (новой) записи блока.

Кроме того, в отдельный файл выводится ключ записи с вершины стека — первой новой записи блока. Последующая обработка этого файла той же процедурой даст первый индексный массив БД, а созданного при этом аналогичного файла — второй индексный массив. При выводе ключ усекается по позиции (сама позиция остается) первого несовпадения с ключом последней занесенной в предыдущий блок записи (см. описание содержания индексного массива в предыдущем разделе).

Далее процесс чтения новых записей и копирования их в выходной массив продолжается. По его завершении цель — формирование необходимой структуры — достигнута.

Сделаем несколько замечаний. Во-первых, если и в стеке вложенных ключей применять сокращение по Куперу, не сокращая запись, помещаемую в пустой стек, то о прерывании процесса сокращения в начале нового блока можно специально не заботиться. Во-вторых, в стек можно не помещать (с соответствующей модификацией алгоритма начала вывода нового блока) записи, ключи которых не могут присоединять окончаний (например, предлоги), о чем подробнее говорилось в конце предыдущего раздела. В-третьих, алгоритм будет правильно обрабатывать случаи, когда в словаре имеются записи с одинаковыми ключами (если это вообще возможно обработать правильно, то есть если группа записей с одинаковыми ключами не превышает по размеру блок).

Алгоритм распаковки очевиден. Перебирая блоки словаря с первого и далее, в каждом из них разворачиваем сокращенные по Куперу записи и переписываем их на выход. Пробелы, заполняющие неиспользованное пространство в конце блока, игнорируем. На выход не переписываются также и первые, «лишние» записи блока, число которых мы ранее поместили в заголовок блока. Эти записи, впрочем, использовались в процессе восстановления по Куперу.

Можно предложить различные варианты алгоритма поиска в таком словаре и различные формы организации его индексного массива [8]. В выполненной автором реализации хранящийся в памяти индексный массив составлен из ключей первых записей блоков (с учетом сделанного в предыдущем разделе замечания). Сам этот массив имеет в точности такую же организацию, как и основной массив, однако тела его записей пусты. Он так же разбит на блоки, и по ним составлен еще один такой же индексный массив, объем которого меньше размера одного блока. При поиске одной и той же процедурой сначала просматривается индексный массив второго уровня, затем соответствующий блок индексного массива первого уровня, а затем соответствующий блок основного файла, вызванный для этой цели с диска.

При поиске в блоке ключи его записей последовательно сравниваются с предъявленной цепочкой (сокращение по Куперу только ускоряет процедуру сравнения). Позиции записей, ключи которых вкладываются слева в нее, заносятся в стек. Поиск прекращается, когда блок исчерпан либо найдена запись с ключом, лексикографически большим предъявленной цепочки. Затем из стека выбираются позиции найденных записей (а при поиске в индексном массиве лишь возвращается номер записи, на которой процесс остановился. Стек в этом случае можно не поддерживать).

Поскольку время работы процедуры просмотра блока существенно меньше времени ожидания считывания блока с диска, автор счел возможным не оптимизировать ее. В случае же, если это необходимо, можно добавить в каждый блок небольшой индексный массив, составленный, скажем, по каждой шестой его записи (если в блоке помещается порядка N записей, то индексный массив стоит делать порядка $\sqrt{N}$), с соответствующим прерыванием сжатия по Куперу.

Относительно слабым местом обсуждаемой структуры словаря является его обновление — добавление новых слов (удаление и изменение производятся

аналогично или проще). При добавлении большого количества слов или если время работы алгоритма актуализации словаря несущественно (в пределах пяти — пятнадцати минут), проще всего обойтись распаковкой имеющейся БД в текстовый вид, внесением в него всех изменений (слияние с отсортированным массивом новых записей) и повторной записью во внутренний вид. Соответствующие алгоритмы, как мы видели выше, имеют достаточное быстродействие.

Однако если нужно быстро внести в словарь на диске одно или несколько новых слов, приходится применять специальные ухищрения. Для этого необходимо, например, заранее при формировании словаря оставлять пустыми некоторые блоки (например, каждый двадцатый). При внесении новых слов весь отрезок словаря, начиная с блока, содержащего алфавитное место первой из новых записей, должен переформатироваться работающими «по конвейеру» алгоритмами распаковки, слияния с новыми записями и упаковки. Переформатирование заканчивается, например, на блоке, вид которого не изменился в данном процессе; если же не все новые записи исчерпаны, то процесс повторяется.

То, что участок, подвергаемый переформатированию, в случае внесения одного слова будет иметь небольшую протяженность, гарантируется двумя соображениями. Во-первых, заранее оставленные пустыми блоки компенсируют рост объема словаря и поглотят волну перемещения его записей. Во-вторых, блоки, в ключ первой записи которых не вкладывается слева ключ новой записи, не должны подвергаться изменениям, кроме как вследствие раздвижки словаря. Время работы такой процедуры не должно превышать в среднем нескольких секунд.

Конечно, необходимо предусмотреть утилиту полного переформатирования словаря, когда пустых блоков станет слишком мало. Кроме того, возникают обычные сложности с нарушением структуры БД при сбоях аппаратуры во время ее обновления, однако их можно обойти применением обычных же алгоритмов сохранения изменяемых блоков на время транзакции БД, а также путем ведения журнала изменений.

ЗАКЛЮЧЕНИЕ

Рассмотрена задача: найти все имеющиеся в БД записи, ключи которых являются начальными подцепочками предъявленной буквенной цепочки. Ни одна из известных автору существующих СУБД не ориентирована на такую задачу. Показано, что дублирование в начало каждого блока всех записей, ключи которых вкладываются слева в ключ первой его записи, гарантирует нахождение всех необходимых записей в первом же блоке, считанном алгоритмом с диска. Увеличение объема словаря — около 10% — можно считать приемлемым. Возможное же некоторое увеличение вычислительной сложности алгоритмов доступа к БД можно считать оправданным, поскольку время их работы существенно меньше времени одного обращения к дисковой памяти, так что минимизации подлежит в первую очередь количество таких обращений.

Обсуждались также алгоритмы формирования внутреннего вида БД, ее распаковки и пополнения. Алгоритмы формирования/распаковки являются однопроходными. Кроме того, они не требуют лингвистической информации, в

частности, не требуют сведений о системе окончаний словоформ данного языка и о его грамматике.

Процедуры поиска, формирования, распаковки и актуализации БД, основанные на предложенных принципах, могут быть реализованы и отдельно, вне связи с конкретной системой морфоанализа.

Список литературы

1. Апресян Ю.Д. и др. Лингвистическое обеспечение системы автоматического перевода ЭТАП-2. — М.: Наука. — 1989. — 296 с.
2. Белоногов Г.Г., Зеленков Ю.Г. Алгоритм морфологического анализа русских слов // Вопр. информ. теории и практики. — М.: ВИНИТИ. — 1985, N 53. — С. 62–93.
3. Герр Р.Г. В поисках золотой середины // Мир ПК. — 1990. — N 5. — С. 43–50.
4. Гельбух А.Ф. Модель морфологии флективного естественного языка // Материалы III Международной конференции «Программное обеспечение ЭВМ», секция 4. — Тверь: НПО Центрпрограммсистем, 1990. — С. 27-–30.
5. Мартин Дж. Организация баз данных в вычислительных системах. — М.: Мир. — 1980. — 660 с.
6. Чижов А. А. Системные программные средства ПЭВМ. — М.: Финансы и статистика, 1990. — 350 с.
7. Большаков И.А. Количественный анализ методов сжатия крупных машинных морфологических словарей // НТИ. Сер. 2. — 1990. — N 3. — С. 28–32.
8. Кнут Д. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. — М: Мир. — 1978. С. 563–570.