УДК 004.043 DOI: 10.31114/2078-7707-2022-1-9-13

Методика эффективного построения таблиц больших размеров

А.А. Лялинский

Институт проблем проектирования в микроэлектронике РАН, г. Москва

zelyal@inbox.ru

***Аннотация* — Исследуются вопросы построения таблиц данных больших (от сотен тысяч записей и выше) размеров. Показаны недостатки применения общепринятых подходов для решения этой задачи. Предложен алгоритм, существенно повышающий скорость обработки данных такого размера.**

***Ключевые слова* — таблицы имен, хранение данных, обработка текстовых файлов, линейный поиск, бинарный поиск, алгоритмы сортировки.**

I. Введение

В ряде задач, например при чтении файлов больших размеров, возникает проблема: каким образом сохранять получаемую из разбора семантическую информацию так, чтобы она занимала как можно меньше места и позволяла быстрый последующий поиск по месту хранения? Наиболее естественный в такого рода задачах объект хранения – имена. Числа обычно не являются самостоятельным предметом поиска, а являются атрибутом какого-либо более значимого объекта. К тому же их фиксированный размер хранения и естественное упорядочение позволяют, при необходимости, применять различного рода ухищрения для ускорения поиска.

Чаще всего в качестве исходного мы имеем набор различающихся по длине имен, причем часть имен может повторяться. Представляет интерес работа с множествами большого размера – от 100 000 и выше. Далее на примерах будет показано, что множества меньших размеров эффективно обрабатываются достаточно простыми алгоритмами за единицы секунд.

Итак, как было упомянуто выше, идеальная методика построения таблицы должна одновременно решать две задачи: обеспечить компактное хранение и последующий быстрый поиск. Добавим сюда еще две задачи – уметь достаточно быстро строить саму таблицу и отсеивать повторяющиеся имена. Рассмотрим, каким образом можно решить эти задачи.

1. Наиболее **компактная** форма **хранения данных** – это хранение элементов данных одним массивом, без каких-либо связующих полей (ссылок) между ними. Такое возможно, если мы заранее знаем количество элементов хранения. Далее будет показано, как снять кажущуюся невозможность получения такой информации.

2-3. **Построение таблицы** за минимально возможное время и оставление в таблице только **уникальных имен** будут рассмотрены в основной части статьи.

4. **Обеспечение быстрого доступа** к данным в таблице. Представляется, что в данном случае наиболее эффективным будет применение двоичного поиска [1-4], который, в свою очередь, требует упорядочивания элементов таблицы по какому-либо критерию. Достаточно часто приемлемым вариантом является упорядочивание по алфавиту.

Далее в работе рассмотрим достаточно естественные способы решения поставленных задач, оценим их производительность, и обоснуем предложение нового подхода, позволяющего существенно сократить временные затраты.

II. Хранение данных

Как сказано выше, наиболее компактный способ хранения данных – это массив элементов без каких-либо связок между ними, что позволяет обращаться к каждому элементу по его индексу. Для этого надо знать точное число элементов или хотя бы верхнюю границу этого множества.

В большинстве случае определить это значение не представляет труда. Действительно, если мы читаем данные из файла, то, учитывая, что любой файл имеет конечный размер, то, скорей всего, можно провести предварительный упрощенный анализ файла, при котором нам не нужно хранить сами данные, а нужно только выделить семантические лексемы с целью подсчета их общего числа. Это достаточно быстрая задача даже для файлов больших размеров, ибо буферизация операций ввода-вывода в современных операционных системах снимает проблему долгого доступа к записям файла.

Проблема может возникнуть, если данные не только берутся из файла, но и возникают в результате работы рекурсивного или итерационного алгоритма при чтении файла. Но и в этом случае можно выполнить имитацию работы этого алгоритма таким образом, чтобы, не затрагивая сути его работы, только пройти по всем его этапам и оценить количество новых возникающих имен. Другое решение – вывести создаваемые имена в промежуточный файл с отдельным запоминанием их количества.

Подчеркнем, что дальнейшие рекомендации по построению эффективных алгоритмов основаны на хранении данных только в виде массива. При хранении больших объемов данных в виде связных списков большая часть времени уйдет на переходы по указателям, что практически сводит на нет вопросы эффективности алгоритма.

III. Обозначения и стандартный подход к решению задачи

Далее будем использовать следующие обозначения:

- S - исходное неупорядоченное множество имен,
- F - конечное упорядоченное множество имен, |F| <= |S|,
- M - промежуточное неупорядоченное множество.

Стандартный подход к решению задачи предполагает прохождение следующих трех этапов (рис. 1) – выборка элемента, проверка его существования в конечном множестве, и внесение его в конечное множество.

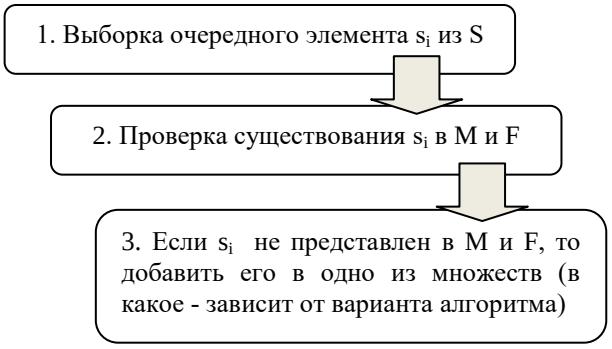

Рис. 1. Стандартный подход к решению задачи построения таблицы

Здесь S и M – неупорядоченные множества, допускающие только линейный поиск перебором; F – упорядоченное множество, по которому можно вести бинарный поиск.

Следует отметить, что во всех описываемых далее вариантах метода построения таблиц подход к использованию сортировки классифицируется как *неустойчивый*, т.е. по мере занесения очередных элементов в конечный массив ранее полученные адреса элементов меняются. Это вынуждает по завершению полного формирования конечного множества F выполнять операцию поиска точного (*устойчивого*) адреса для каждого элемента таблицы. Заметим, что этот этап операции очень эффективно выполняется с использованием алгоритмов распараллеливания, например, распараллеливание на основе директив OMP-библиотеки (*Open Multi-Processing*) [5], идеально подходящих для решения этого задачи. В таблицах ниже это время не включено, так как оно одинаково для всех вариантов метода и сравнения алгоритмов между собой значения не имеет.

IV. Два простейших варианта

Вначале рассмотрим два крайних подхода, для множеств больших размеров не имеющих практической ценности.

Вариант 1: массив M не используется, а сортировка множества F проводится сразу же после добавления в него очередного элемента s_i. Недостаток метода: суммарные затраты на сортировку (на этапе 3) слишком велики, особенно при выборке последних элементов исходного множества S, когда размер конечного множества F становится большим.

Вариант 2: элементы s_i постоянно добавляются в промежуточное множество M, сортировка проводится только после выборки всех элементов из S. Недостаток метода: накопление больших временных затрат на этапе проверки (этап 2), так как поиск приходится вести простым перебором на неупорядоченном множестве.

Для оценки временных затрат в описываемых методах была подготовлена тестовая программа. В нев вначале формируется множество S указанного размера, при этом длина имен варьируется случайным образом от 1 до 25, содержимое имен также формируется случайным образом из фиксированного набора символов. Для большего приближения к реальным задачам около 70% имен формируются уникальными, а остальная часть – повтором существующих (это важно для того, чтобы имеющиеся в алгоритмах операции поиска давали ненулевой эффект). В табл. 1 приведены результаты прогонов для диапазона количества имен от 500 до 500 000. Пустые клетки говорят о том, что тестирование в этих позициях не имело смысл, так как налицо было стремление алгоритма к огромным временным затратам.

Сравнивая результаты работы этих двух методов можно сделать несколько неожиданный промежуточный вывод, что сумма затрат на сортировку (вариант 1) существенно больше сумм затрат по линейный поиск (вариант 2).

Таблица 1

Временные затраты с использованием методов 1 и 2

Вариант	Количество имен в исходной таблице						
	500	1 000	5 000	10 000	50 000	100 000	500 000
	Временные затраты, сек						
1	0.006	0.020	0.55	2	63		
2	0.001	0.002	0.030	0.13	2.7	16	712

V. Методы с промежуточным буфером

Исходя из вышеизложенного представляется логичным рассмотреть варианты, представляющие собой комбинацию первого и второго методов: сортировка проводится по мере достижения множеством M некоторого размера или по иным критериям.

Для лучшего понимания представим алгоритм в более подробном виде (см. рис. 2). Представляющие интерес блоки помечены цифрами. Для простоты не показано, что алгоритм завершает работу после выборки всех элементов из S. Наиболее затратными являются блоки 1, 2 и 4. В блоке 1 проводится линейный поиск элемента s_i по неупорядоченному множеству M. В блоке 2 проводится бинарный поиск элемента s_i по отсортированному множеству F. В блоке 4 выполняется сортировка обновленного множества F после добавления в него элементов множества M.

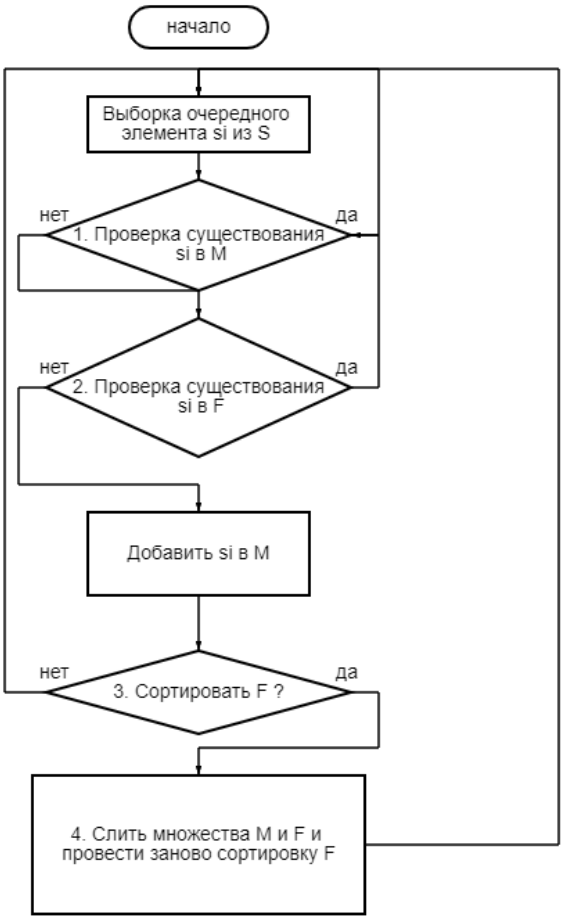

Рис. 2. Более детальное представление методов с промежуточным буфером

<u>В варианте 3</u> решение о сортировке (блок 3) принимается по мере наполнения множества M. Как только его размер превышает некое заранее заданное число, то принимается решение о слиянии множеств M и F и последующей сортировке F (и очищении M).

<u>Вариант 4</u> отличается от варианта 3 тем, что блоки 1 и 2 поменяны местами – вначале выполняется бинарный поиск, затем линейный.

<u>Вариант 5</u> отличается от предыдущих введением элемента адаптивности. Времена выполнения блоков 1 и 2 суммируются отдельно для каждого. После выборки элемента решение о том, где вначале провести поиск – в блоке 1 или 2 – принимается в зависимости от текущих суммарных временных затрат на эти блоки. Решение о сортировке (блок 3) принимается в том случае, если размер множества M превысил заранее заданное число. Тогда он присоединяется к F, который заново сортируется, а M очищается.

Параметром данной группы методов является размер промежуточного множества M. В данной группе тестов он задается в процентах от размера исходного множества. В табл. 2-5 представлены результаты прогона тестов. Так как в методах этой группы мы не сразу получаем истинный адрес имени, то для чистоты эксперимента по завершению построению таблицы добавлен формальный этап, на котором проводится поиск каждого элемента таблицы в самой таблице. На практике этот этап необходим для получения истинного адреса каждого имени.

Таблица 2

Временные затраты с использованием методов 3-5 (предел буфера – 0.5%)

Метод	Количество имен в исходной таблице			
	100 000	500 000	750 000	1 000 000
	Временные затраты, сек			
3	0.7	8	14	24
4	0.7	7	13	22
5	0.9	8	15	22

Таблица 3

Временные затраты с использованием методов 3-5 (предел буфера – 2.5%)

Метод	Количество имен в исходной таблице			
	100 000	500 000	750 000	1 000 000
	Временные затраты, сек			
3	0.6	12	27	49
4	0.4	10	21	39
5				

Таблица 4

Временные затраты с использованием методов 3-5 (предел буфера – 5%)

Метод	Количество имен в исходной таблице			
	100 000	500 000	750 000	1 000 000
	Временные затраты, сек			
3	1	24	53	107
4	0.7	17	40	86
5				

Таблица 5

Временные затраты с использованием методов 3-5 (предел буфера – 10%)

Метод	Количество имен в исходной таблице			
	100 000	500 000	750 000	1 000 000
	Временные затраты, сек			
3	1.5	50	157	316
4	1.3	39	117	241
5	1.3	39	123	255

Мы видим, что сами варианты методов 3-5 дают достаточно близкие результаты, особенно при начальных значениях размеров исходного множества. Гораздо большее влияние на скорость работы оказывает размер промежуточного буфера – чем он меньше, тем выше скорость работы. Очевидно, что решающим фактором здесь оказывается время поиска по неупорядоченному множеству – чем больше буфер, тем больше затраты в целом. Возможно, имеет смысл определять его размер не в процентах от размера множества, а неким фиксированным числом.

VI. Беспоисковый метод

Анализ предыдущих методов говорит о том, что при работе с множествами размером выше миллиона скорей всего получить решение за приемлемое время не удастся. К тому же выбор размера промежуточного буфера требует отдельной настройки. Попробуем применить другой подход.

Заметим, что большая часть исходного множества – это уникальные имена. Все предыдущие алгоритмы тратят значительные усилия на то, чтобы убедится, что очередное имя отсутствует в формируемой таблице имен, о чем, на самом деле, нам заранее известно. Поэтому откажемся от проверки имен на существование совсем. Тогда алгоритм принимает достаточно простой вид (рис. 3):

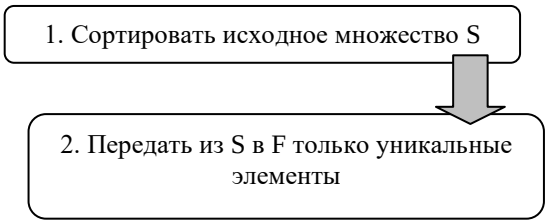

Рис. 3. Алгоритм беспоискового метода

Уникальность элементов в отсортированном массиве определяется достаточно просто – все одинаковые элементы в нем расположены по соседству. Поэтому мы просто двигаемся подряд по всем элементам отсортированного множества и если элемент совпадает с последним просмотренным, то пропускаем его. В табл. 6 приведены результаты прогонки данного метода на тестах с размером от 100 000 до 100 миллионов. Можно констатировать чуть большую, чем линейную, зависимость времени работы от размера множества. Так, например, увеличение размеров исходного множества на порядок приводит почти к такому росту расходов.

Таблица 6

Временные затраты с использованием метода 6

Варианты метода	Количество имен в исходной таблице (миллионов)										
	0.1	0.5	0.75	1	2,5	5	10	25	50	75	100
	Временные затраты, сек										
сортировка и очистка	0.03	0.18	0.3	0.4	1.3	2.8	6.7	19	46	76	107
то же + выборка	0.04	0.25	0.42	0.62	1.7	3.6	8.8	24	61	94	127
то же + парал. выборка	0.03	0.22	0.33	0.47	1.3	3.6	7.4	22	49	81	112

Пояснения относительно трех последних строк таблицы. В строке «сортировка и очистка» приведены затраты только на этапы сортировки и удаления лишних элементов. В строке «то же + выборка» к ним добавлены затраты на формальный этап поиска каждого элемента новой таблицы в ней самой. В строке «то же + парал. выборка» этот этап поиска выполнен с применением алгоритмов распараллеливания на основе директив OMP-библиотеки (*Open Multi-Processing).*

VII. Выводы

Проведен анализ стандартных методов построения таблиц хранения больших размеров. Показаны их ограниченность при применении к множествам сверх-больших размеров. Предложен более простой и эффективный метод, позволяющий обрабатывать множества в десятки и сотни миллионов объектов за время порядка нескольких минут. Метод может быть легко реализован в различного рода задачах САПР микроэлектроники [6-11].

Литература

[1] Д. Кнут: Искусство программирования. Том 3. Сортировка и поиск. 1973. first edition ISBN 0-201-03803-X.

[2] Обзор алгоритмов сортировки на Википедии: https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8 (дата проверки: 04.04.2022)

[3] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ = INTRODUCTION TO ALGORITHMS. — 2-е изд. — М.: «Вильямс», 2006. — С. 1296. — ISBN 5-8459-0857-4.

[4] Роберт Седжвик. Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск = Algorithms in C. Fundamentals/Data Structures/Sorting/Searching. — СПб.: ДиаСофтЮП, 2003. — С. 672. — ISBN 5-93772-081-4.
[5] Официальный сайт OpenMP https://www.openmp.org/ (дата проверки: 04.04.2022)
[6] Гаврилов С.В., Железников Д.А., Чочаев Р., Хватов В.М. Алгоритм декомпозиции на основе метода имитации отжига для реконфигурируемых систем на кристалле // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2018. Выпуск 1. С. 199-204. doi:10.31114/2078-7707-2018-1-199-204
[7] Фролова П.И., Чочаев Р., Иванова Г.А., Гаврилов С.В. Алгоритм размещения с оптимизацией быстродействия на основе матриц задержек для реконфигурируемых систем на кристалле // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2020. Выпуск 1. С. 2-7. doi:10.31114/2078-7707-2020-1-2-7
[8] Иванников А.Д., Стемпковский А.Л. Анализ итерационных методов решения систем логических уравнений и их использование при моделировании цифровых систем // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2020. Выпуск 3. С. 2-8. doi:10.31114/2078-7707-2020-3-2-8
[9] Заплетина М.А., Железников Д.А., Гаврилов С.В. Иерархический подход к трассировке реконфигурируемой системы на кристалле островного типа // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2020. Выпуск 3. С. 16-21. doi:10.31114/2078-7707-2020-3-16-21
[10] Лялинский А.А. ПЛИС-ориентированная библиотека логических функций // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2020. Выпуск 1. С. 15-19. doi:10.31114/2078-7707-2020-1-15-19
[11] Лялинский А.А. Характеризация библиотек цифровых схем с использованием веб-технологий // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2020. Выпуск 2. С. 29-34. doi:10.31114/2078-7707-2020-2-29-34.

Methodology for the Effective Construction of Large Tables

A.A. Lyalinsky

Institute for Design Problems in Microelectronics
of Russian Academy of Sciences (IPPM RAS)

zelyal@inbox.ru

***Abstract* — The issues of constructing data tables of large (from hundreds of thousands of records and above) sizes are investigated. The disadvantages of using generally accepted approaches to solve this problem are shown. An algorithm is proposed that significantly increases the processing speed of data of this size.**

***Keywords* — name tables, data storage, text file processing, linear search, binary search, sorting algorithms.**

REFERENCES

[1] Donald E. Knuth, Sorting and Searching, Second Edition. Reading, Massachusetts: Addison-Wesley, 1998, 780pp. ISBN 0-201-89685-0
[2] Survey sorting algorithms in Wikipedia: https://en.wikipedia.org/wiki/Sorting_algorithm (checking date: 04.04.2022)
[3] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009), "8", Introduction To Algorithms (3rd ed.), Cambridge, MA: The MIT Press, p. 167, ISBN 978-0-262-03293-3
[4] Sedgewick, Robert (1 September 1998). Algorithms In C: Fundamentals, Data Structures, Sorting, Searching, Parts 1-4 (3 ed.). Pearson Education. ISBN 978-81-317-1291-7..
[5] Official site of OpenMP https://www.openmp.org/ (checking date: 04.04.2022)
[6] Gavrilov S.V., Zheleznikov D.A., Chochaev R., Khvatov V.M. Partitioning Algorithm Based on Simulated Annealing for Reconfigurable Systems-on-Chip // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2018. Issue 1. P. 199-204. doi:10.31114/2078-7707-2018-1-199-204
[7] Frolova P.I., Chochaev R., Ivanova G.A., Gavrilov S.V. Timing-driven placement algorithm based on delay matrix model for reconfigurable system-on-chip // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2020. Issue 1. P. 2-7. doi:10.31114/2078-7707-2020-1-2-7
[8] Ivannikov A.D., Stempkovsky A.L. Analysis of Iterative Methods for Solving Logical Equation Systems and their Use in Digital System Simulation // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2020. Issue 3. P. 2-8. doi:10.31114/2078-7707-2020-3-2-8
[9] Zapletina M.A., Zheleznikov D.A., Gavrilov S.V. The Hierarchical Approach to Island Style Reconfigurable System-on-a-chip Routing // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2020. Issue 3. P. 16-21. doi:10.31114/2078-7707-2020-3-16-21
[10] Lyalinsky A.A. Characterization of FPGA-based Digital Libraries // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2020. Issue 1. P. 15-19. doi:10.31114/2078-7707-2020-1-15-19
[11] Lyalinsky A.A. Web-based Characterization of Digital Libraries // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2020. Issue 2. P. 29-34. doi:10.31114/2078-7707-2020-2-29-34.